

Oxygen XML Editor 17.0

Notice

Copyright

Oxygen XML Editor User Manual

Syncro Soft SRL.

Copyright © 2002-2015 Syncro Soft SRL. All Rights Reserved.

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher. Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

Trademarks. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and Syncro Soft SRL was aware of a trademark claim, the designations have been rendered in caps or initial caps.

Notice. While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Link disclaimer. Syncro Soft SRL is not responsible for the contents or reliability of any linked Web sites referenced elsewhere within this documentation, and Syncro Soft SRL does not necessarily endorse the products, services, or information described or offered within them. We cannot guarantee that these links will work all the time and we have no control over the availability of the linked pages.

Warranty. Syncro Soft SRL provides a limited warranty on this product. Refer to your sales agreement to establish the terms of the limited warranty. In addition, Oxygen XML Editor End User License Agreement, as well as information regarding support for this product, while under warranty, is available through the [Oxygen XML Editor website](#).

Third-party components. Certain software programs or portions thereof included in the Product may contain software distributed under third party agreements ("Third Party Components"), which may contain terms that expand or limit rights to use certain portions of the Product ("Third Party Terms"). Information identifying Third Party Components and the Third Party Terms that apply to them is available on the [Oxygen XML Editor website](#).

Downloading documents. For the most current versions of documentation, see the [Oxygen XML Editor website](#).

Contact Syncro Soft SRL. Syncro Soft SRL provides telephone numbers and e-mail addresses for you to report problems or to ask questions about your product, see the [Oxygen XML Editor website](#).

Contents

Chapter 1: Introduction.....21

Chapter 2: Installation.....25

Installation Options.....	26
Windows Installation.....	26
Mac OS X Installation.....	28
Linux Installation.....	29
Windows Terminal Server Installation.....	31
Linux Server Installation.....	33
Java Web Start (JWS) Installation.....	34
Site-wide deployment.....	36
Licensing.....	36
Setting up a License Server.....	40
Setting up a Floating License Server Running as a Standalone Process Using a Platform-independent Distribution.....	44
Transferring or Releasing a License.....	45
Upgrading.....	45
Installing and Updating Add-ons.....	46
Uninstalling.....	47
Installer Command Line Reference.....	47

Chapter 3: Getting Started.....51

Your First XML Document.....	52
------------------------------	----

Chapter 4: DITA Authoring Guide.....57

Your First DITA Topic.....	58
Topics and Topic Structure.....	62
Editing DITA Topics.....	63
Adding Images to a DITA Topic.....	65
Adding Tables to a DITA Topic.....	67
Documents and Maps.....	69
Adding Topics to a DITA Map.....	69
Creating DITA Sub-Maps.....	70
Chunking DITA Topics.....	70
Manage a DITA Map.....	71
Creating a Book in DITA.....	71
Creating a Table of Contents in DITA.....	72

Creating an Index in DITA.....	72
Validate a DITA Map.....	73
Reuse.....	73
Reusing DITA Topics.....	74
Creating a DITA Content Reference.....	74
Creating a DITA Content Key Reference.....	74
Creating a Reusable Content Component.....	75
Insert a Reusable Content Component.....	76
Profiling (Conditional Content) in DITA.....	76
Creating Variable Text in DITA.....	78
Linking.....	79
Hierarchical Linking in DITA.....	80
Linking with Relationship Tables in DITA.....	80
Linking with a <code>related-links</code> Element in DITA.....	81
Inline Linking in DITA.....	82
Output.....	82
Generating Output from DITA Content.....	83
Metadata.....	83
Background: Keys.....	84
Chapter 5: Perspectives.....	85
Perspectives.....	86
Editor Perspective	86
XSLT Debugger Perspective	89
XQuery Debugger Perspective	90
Database Perspective	90
Dockable Views and Editors.....	91
Help Menu.....	93
The About Dialog.....	94
The Welcome Dialog.....	94
Chapter 6: Editing Modes.....	97
Text Editing Mode.....	98
The Undo/Redo Actions.....	98
Copying and Pasting Text.....	98
Finding and Replacing Text in the Current File.....	98
Finding and Replacing Text in Multiple Files.....	102
Changing the Font Size.....	105
Word/Line Editor Actions.....	105
Dragging and Dropping the Selected Text.....	106
Inserting a File at Caret Position.....	106
Opening the File at Caret in System Application.....	106
Opening the File at Caret Position.....	106

Printing a File.....	106
Bidirectional Text Support in Text Mode.....	107
Grid Editing Mode.....	108
Layouts: Grid and Tree.....	109
Grid Move Navigation.....	109
Specific Grid Actions.....	110
Drag and Drop in the Grid Editor.....	111
Copy and Paste in the Grid Editor.....	111
Bidirectional Text Support in Grid Mode.....	113
Author Editing Mode.....	113
Tagless XML Authoring.....	114
General Author Presentation.....	117
Smart Paste Support.....	166
Bidirectional Text Support in Author Mode.....	167

Chapter 7: Editing Documents.....171

Working with Unicode.....	172
Opening and Saving Unicode Documents.....	172
Inserting Symbols.....	173
Creating, Opening, and Closing Documents.....	174
Creating Documents.....	174
Saving Documents.....	178
Opening/Navigating Documents.....	178
Opening and Saving Remote Documents via FTP/SFTP/WebDAV/SharePoint	186
Opening the Current Document in System Application.....	191
Switching Between Opened Tabs.....	191
Closing Documents.....	191
The Contextual Menu of the Editor Tab.....	192
Viewing File Properties.....	192
Grouping Documents in XML Projects.....	193
Using the Project View.....	193
Defining Master Files at Project Level.....	200
Editing XML Documents.....	203
Associate a Schema to a Document.....	203
Content Completion Assistant.....	207
Validating XML Documents.....	215
Document Navigation.....	224
Large Documents.....	229
Working with XML Catalogs.....	231
XML Resource Hierarchy/Dependencies View.....	233
Converting Between Schema Languages.....	235
Editing XML Tree Nodes.....	237
Formatting and Indenting XML Documents.....	237
Editing Modular XML Files in the Master Files Context.....	242

Managing ID/IDREFS.....	242
Search and Refactor Operations Scope.....	244
Viewing Status Information.....	245
Image Preview.....	245
Making a Persistent Copy of Results.....	245
Locking and Unlocking XML Markup.....	246
Adjusting the Transparency of XML Markup.....	246
XML Editor Specific Actions.....	247
XML Quick Fixes.....	252
Refactoring XML Documents.....	253
Editing XSLT Stylesheets.....	264
Validating XSLT Stylesheets.....	265
Editing XSLT Stylesheets in the Master Files Context.....	265
Syntax Highlight.....	266
Content Completion in XSLT Stylesheets.....	266
The XSLT/XQuery Input View.....	270
The XSLT Outline View.....	272
XSLT Stylesheet Documentation Support.....	274
Generating Documentation for an XSLT Stylesheet.....	275
Finding XSLT References and Declarations.....	282
Highlight Component Occurrences.....	283
XSLT Refactoring Actions.....	283
XSLT Resource Hierarchy/Dependencies View.....	285
Component Dependencies View.....	288
XSLT Quick Assist Support.....	289
XSLT Quick Fix Support	290
Linking Between Development and Authoring.....	292
XSLT Unit Test (XSpec).....	292
Editing Ant Build Files.....	294
Validate Ant Build Files.....	294
Editing Ant Build Files in the Master Files Context.....	294
Syntax Highlight.....	295
Content Completion in Ant Build Files.....	295
Ant Outline View.....	295
Find References and Declarations of Ant Components.....	297
Highlight Component Occurrences.....	297
Ant Refactoring Actions.....	298
Ant Resource Hierarchy/Dependencies View.....	298
Ant Component Dependencies View.....	299
Ant Quick Assist Support.....	300
Ant Quick Fix Support.....	301
Editing XML Schemas.....	301
XML Schema Diagram Editing Mode.....	301
XML Schema Text Editing Mode.....	334
Editing XML Schema in the Master Files Context.....	336

Searching and Refactoring Actions in XML Schemas.....	336
Component Dependencies View.....	337
XML Schema Quick Assist Support.....	339
XML Schema Resource Hierarchy / Dependencies View.....	339
Generating Documentation for an XML Schema.....	342
Flatten an XML Schema.....	349
Generate Sample XML Files.....	351
XML Schema Regular Expressions Builder.....	355
Create an XML Schema From a Relational Database Table.....	357
XML Schema 1.1.....	357
Setting the XML Schema Version.....	358
Linking Between Development and Authoring.....	359
Editing XQuery Documents.....	359
XQuery Outline View.....	359
Folding in XQuery Documents.....	360
Formatting and Indenting XQuery Documents.....	361
Generating HTML Documentation for an XQuery Document.....	361
Editing WSDL Documents.....	362
WSDL Outline View.....	362
Content Completion in WSDL Documents.....	365
Editing WSDL Documents in the Master Files Context.....	366
Searching and Refactoring Operations in WSDL Documents.....	367
Searching and Refactoring Operations Scope in WSDL Documents.....	368
WSDL Resource Hierarchy/Dependencies View in WSDL Documents.....	369
Component Dependencies View in WSDL Documents.....	372
Highlight Component Occurrences in WSDL Documents.....	373
Quick Assist Support in WSDL Documents.....	373
Generating Documentation for WSDL Documents.....	374
WSDL SOAP Analyzer.....	377
Editing CSS Stylesheets.....	380
Validating CSS Stylesheets.....	380
Content Completion in CSS Stylesheets.....	381
CSS Outline View.....	381
Folding in CSS Stylesheets.....	382
Formatting and Indenting CSS Stylesheets (Pretty Print).....	382
Minifying CSS Stylesheets.....	382
Other CSS Editing Actions.....	383
Editing LESS CSS Stylesheets.....	383
Validating LESS Stylesheets.....	383
Content Completion in LESS Stylesheets.....	383
Compiling LESS Stylesheets to CSS.....	384
Editing Relax NG Schemas.....	384
Editing Relax NG Schema in the Master Files Context.....	384
Relax NG Schema Diagram.....	385
Relax NG Editor Specific Actions.....	389

Searching and Refactoring Actions in RNG Schemas.....	389
RNG Resource Hierarchy/Dependencies View.....	390
Component Dependencies View.....	393
RNG Quick Assist Support.....	394
Configuring a Custom Datatype Library for a RELAX NG Schema.....	394
Linking Between Development and Authoring.....	395
Editing NVDL Schemas.....	395
NVDL Schema Diagram.....	395
NVDL Editor Specific Actions.....	397
Searching and Refactoring Actions in NVDL Schemas.....	397
Component Dependencies View.....	398
Linking Between Development and Authoring.....	399
Editing JSON Documents.....	399
JSON Editor Text Mode.....	399
JSON Editor Grid Mode.....	401
JSON Outline View.....	402
Validating JSON Documents.....	402
Convert XML to JSON.....	402
Editing StratML Documents.....	403
Editing JavaScript Documents.....	404
JavaScript Editor Text Mode.....	404
Content Completion in JavaScript Files.....	406
JavaScript Outline View.....	406
Validating JavaScript Files.....	407
Editing XProc Scripts.....	407
Editing Schematron Schemas.....	408
Validate an XML Document.....	409
Validating Schematron Documents.....	409
Content Completion in Schematron Documents.....	409
RELAX NG/XML Schema with Embedded Schematron Rules.....	410
Editing Schematron Schema in the Master Files Context.....	411
Schematron Resource Hierarchy/Dependencies View.....	411
Highlight Component Occurrences in Schematron Documents.....	413
Searching and Refactoring Operations in Schematron Documents.....	413
Searching and Refactoring Operations Scope in Schematron Documents.....	414
Quick Assist Support in Schematron Documents.....	415
Editing Schematron Quick Fixes.....	416
Validating Schematron Quick Fixes.....	416
Content Completion in SQF.....	416
Highlight Quick Fix Occurrences in SQF.....	416
Searching and Refactoring Operations in SQF.....	417
Embed Schematron Quick Fixes in Relax NG or XML Schema.....	418
Customizing Schematron Quick Fixes.....	418
Editing SVG Documents.....	422
The Standalone SVG Viewer.....	423

The Preview Result Panel.....	423
Editing XHTML Documents.....	424
Spell Checking.....	424
Spell Checking Dictionaries.....	425
Learned Words.....	427
Ignored Words.....	427
Automatic Spell Check.....	427
Spell Checking in Multiple Files.....	427
AutoCorrect Misspelled Words.....	429
Add Dictionaries for the AutoCorrect Feature.....	429
Editing Large Documents.....	430
File sizes smaller than 300 Megabytes.....	430
File sizes greater than 300 MB.....	430
Scratch Buffer.....	431
Handling Read-Only Files.....	431
Editing Documents with Long Lines.....	431
Associating a File Extension with Oxygen XML Editor.....	431

Chapter 8: Author for DITA.....433

Creating DITA Maps and Topics.....	434
DITA Maps Manager.....	434
Creating a Map.....	438
Selecting a Root Map.....	439
Create a Topic in a Map.....	439
Organize Topics in a Map.....	439
Creating Relationship Tables.....	439
Validating DITA Maps.....	440
Finding Resources Not Referenced in DITA Maps.....	441
Insert and Edit References.....	441
Transforming DITA Maps and Topics.....	444
DITA OT Transformation.....	444
DITA Map WYSIWYG Transformation.....	451
Set a Font for PDF Output Generated with Apache FOP.....	452
DITA-OT Customization.....	453
Support for Transformation Customizations.....	453
Using Your Custom Build File.....	453
Customizing the Oxygen XML Editor Ant Tool.....	453
Increasing the Memory for the Ant Process.....	453
Resolving Topic References Through an XML Catalog.....	454
DITA to PDF Output Customization.....	454
Creating a DITA OT Customization Plugin.....	456
Installing a Plugin in the DITA Open Toolkit.....	457
DITA Specialization Support.....	458
Integration of a DITA Specialization.....	458

Editing DITA Map Specializations.....	459
Editing DITA Topic Specializations.....	459
Use an External DITA Open Toolkit in Oxygen XML Editor.....	459
Reusing Content.....	460
Working with Content References.....	460
How to Work with Reusable Components.....	460
Insert a Direct Content Reference.....	461
Moving and Renaming Resources.....	463
DITA Profiling / Conditional Text.....	463
Profiling / Conditional Text Markers.....	464
Profiling with a Subject Scheme Map.....	465
Publish Profiled Text.....	466
How to Profile DITA Content.....	466
Working with MathML.....	466
MathML Equations in the HTML Output.....	467

Chapter 9: Predefined Document Types.....469

Document Type.....	471
The DocBook 4 Document Type.....	471
The DocBook 5 Document Type.....	481
The DITA Topics Document Type.....	488
The DITA Map Document Type.....	497
The XHTML Document Type.....	504
The TEI ODD Document Type.....	507
The TEI P4 Document Type.....	509
The TEI P5 Document Type.....	511
The JATS Document Type.....	514
The EPUB Document Type.....	515
The DocBook Targetset Document Type.....	516

Chapter 10: Authoring Customization.....517

Authoring Customization Guide.....	518
Simple Customization Tutorial.....	518
Advanced Customization Tutorial - Document Type Associations.....	523
CSS Support in Author.....	591
Example Files Listings - The Simple Documentation Framework Files.....	641
Author Component.....	646
Creating and Running Automated Tests.....	662
API Frequently Asked Questions (API FAQ).....	664
Difference Between a Document Type (Framework) and a Plugin Extension.....	664
Dynamically Modify the Content Inserted by the Author.....	665
Split Paragraph on Enter (Instead of Showing Content Completion List).....	665
Impose Custom Options for Authors.....	666

Highlight Content.....	666
How Do I Add My Custom Actions to the Contextual Menu?.....	667
Adding Custom Callouts.....	668
Change the DOCTYPE of an Opened XML Document.....	671
Customize the Default Application Icons for Toolbars/Menus.....	671
Disable Context-Sensitive Menu Items for Custom Author Actions.....	671
Dynamic Open File in Oxygen XML Editor Distributed via JavaWebStart.....	672
Change the Default Track Changes (Review) Author Name.....	673
Multiple Rendering Modes for the Same Author Document.....	673
Obtain a DOM Element from an AuthorNode or AuthorElement.....	674
Print Document Within the Author Component.....	674
Running XSLT or XQuery Transformations.....	674
Use Different Rendering Styles for Entity References, Comments or Processing Instructions.....	674
Insert an Element with all the Required Content.....	677
Obtain the Current Selected Element Using the Author API.....	677
Debugging a Plugin Using the Eclipse Workbench.....	678
Debugging an Oxygen SDK Extension Using the Eclipse Workbench.....	678
Extending the Java Functionality of an Existing Framework (Document Type).....	679
Controlling XML Serialization in the Author Component.....	679
How can I add a custom Outline view for editing XML documents in the Text mode?.....	680
Dynamically Adding Form Controls Using a StylesFilter.....	683
Modifying the XML content on Open.....	684
Modifying the XML content on Save.....	685
Save a new document with a predefined file name pattern.....	685
Auto-generate an ID when a document is opened or created.....	686
Use a custom view with the Oxygen XML Editor distribution.....	687
Chapter 11: Transforming Documents.....	689
Transformation Scenarios.....	690
Defining a New Transformation Scenario.....	690
Configure Transformation Scenario(s) Dialog Box.....	717
Duplicating a Transformation Scenario.....	719
Editing a Transformation Scenario.....	719
Apply Batch Transformations.....	719
Built-in Transformation Scenarios.....	720
Sharing the Transformation Scenarios.....	720
Transformation Scenarios View.....	721
Debugging PDF Transformations.....	724
XSLT Processors.....	724
XSL-FO Processors.....	727
Output Formats.....	730
WebHelp Output.....	731

Chapter 12: Querying Documents.....751

Running XPath Expressions.....	752
What is XPath.....	752
Oxygen XPath Toolbar.....	752
The XPath/XQuery Builder View.....	753
XPath Results View.....	755
Catalogs.....	756
XPath Prefix Mapping.....	757
Working with XQuery.....	757
What is XQuery.....	757
Syntax Highlight and Content Completion.....	757
XQuery Outline View.....	758
The XQuery Input View.....	760
XQuery Validation.....	761
Other XQuery Editing Actions.....	762
Transforming XML Documents Using XQuery.....	762

Chapter 13: Debugging XSLT Stylesheets and XQuery Documents.....767

Overview.....	768
Layout.....	768
Control Toolbar.....	769
Information View.....	772
Multiple Output Documents in XSLT 2.0 and XSLT 3.0.....	781
Working with the XSLT / XQuery Debugger.....	781
Steps in a Typical Debug Process.....	781
Using Breakpoints.....	782
Determining What XSLT / XQuery Expression Generated Particular Output.....	782
Debugging Java Extensions.....	784
Supported Processors for XSLT / XQuery Debugging.....	785

Chapter 14: Performance Profiling of XSLT Stylesheets and XQuery

Documents.....787	
Overview.....	788
Viewing Profiling Information.....	788
Invocation Tree View.....	788
Hotspots View.....	789
Working with XSLT/XQuery Profiler.....	789

Chapter 15: Working with Archives.....791

Browsing and Modifying Archive Structure.....	792
---	-----

Working with EPUB.....	794
Create an EPUB.....	795
Publish to EPUB.....	795
Editing Files From Archives.....	795

Chapter 16: Working with Databases.....797

Relational Database Support.....	798
Configuring Database Data Sources.....	798
Configuring Database Connections.....	798
How to Configure Support For Relational Databases.....	798
Resource Management.....	809
SQL Execution Support.....	815
Native XML Database (NXD) Support.....	817
Configuring Database Data Sources.....	817
Configuring Database Connections.....	818
How to Configure Support for Native XML Databases.....	818
Data Source Explorer View.....	822
XQuery and Databases.....	836
Build Queries with Drag and Drop from the Data Source Explorer View.....	836
XQuery Transformation.....	837
XQuery Database Debugging.....	838
WebDAV Connection.....	840
How to Configure a WebDAV Connection.....	840
WebDAV Connection Actions.....	840
BaseX Support.....	842
Resource Management.....	842
XQuery Execution.....	842

Chapter 17: Importing Data.....845

Introduction.....	846
Import from Database.....	846
Import Table Content as XML Document.....	846
Convert Table Structure to XML Schema.....	848
Import from MS Excel Files.....	848
Import from MS Excel 2007-2010 (.xlsx).....	850
Import from HTML Files.....	850
Import from Text Files.....	850
Import Content Dynamically.....	851

Chapter 18: Content Management System (CMS) Integration.....855

Integration with Documentum (CMS) (deprecated).....	856
Configure Connection to Documentum Server.....	856
Documentum (CMS) Actions in the Data Source Explorer View.....	857

Transformations on DITA Content from Documentum (CMS).....	861
Integration with Microsoft SharePoint.....	861
How to Configure a SharePoint Connection.....	861
The SharePoint Browser View.....	862
SharePoint Connection Actions.....	864
Chapter 19: Tools.....	867
SVN Client.....	868
Main Window.....	868
Getting Started.....	878
Syncro SVN Client Views.....	929
The Revision Graph of a SVN Resource.....	956
Oxygen XML Editor SVN Preferences.....	959
Entering Local Paths and URLs.....	959
Technical Issues.....	960
Tree Editor.....	963
Comparing and Merging Documents.....	963
Directories Comparison.....	964
Files Comparison.....	967
XML Digital Signatures.....	974
Overview.....	975
Canonicalizing Files.....	976
Certificates.....	977
Signing Files.....	977
Verifying the Signature.....	978
Large File Viewer.....	978
Hex Viewer.....	980
Integrating External Tools.....	980
Chapter 20: Extending Oxygen XML Editor with Plugins.....	983
Introduction.....	984
General configuration of an Oxygen XML Editor plugin.....	984
Installation.....	985
Types of plugin extensions.....	985
Workspace Access Plugin Extension.....	986
Option Page Plugin Extension.....	987
Components Validation Plugin Extension.....	987
Custom Protocol Plugin Extension.....	988
Resource Locking Custom Protocol Plugin Extension.....	989
XML Refactoring Operations Plugin Extension.....	989
Open Redirect Plugin Extension.....	989
Targeted URL Stream Handler Plugin Extension.....	989
Lock Handler Factory Plugin Extension.....	991

StylesFilter Plugin Extension.....	991
Plugin Extensions designed to work only in the Text Editing Mode.....	991
How to.....	992
How to Write a CMS Integration Plugin.....	992
How to Write A Custom Protocol Plugin.....	995
How to deploy a plugin as an add-on.....	996
How to Share the Classloader Between a Framework and a Plugin.....	996
Example - A Selection Plugin.....	996
Creating and Running Automated Tests.....	998
Debugging a Plugin Using the Eclipse Workbench.....	999
Disabling a Plugin.....	1000

Chapter 21: Configuring Oxygen XML Editor.....1001

Preferences.....	1002
Global Preferences.....	1002
Appearance Preferences.....	1004
Add-ons Preferences.....	1004
Fonts Preferences.....	1004
Document Type Association Preferences.....	1005
Application Layout Preferences.....	1015
Encoding Preferences.....	1015
Editor Preferences.....	1016
CSS Validator Preferences.....	1045
XML Preferences.....	1045
DITA Preferences.....	1064
Data Sources Preferences.....	1065
SVN Preferences.....	1068
Diff Preferences.....	1072
Archive Preferences.....	1074
Plugins Preferences.....	1075
External Tools Preferences.....	1076
Menu Shortcut Keys Preferences.....	1077
File Types Preferences.....	1078
The Open/Find Resources Preferences Page.....	1079
Custom Editor Variables Preferences.....	1080
Network Connection Settings Preferences.....	1080
XML Structure Outline Preferences.....	1083
Views Preferences.....	1083
Messages Preferences.....	1083
Importing / Exporting Global Options.....	1084
Project Level Options.....	1084
Reset Global Options.....	1085
Customizing Default Options.....	1085
Scenarios Management.....	1086

Editor Variables.....	1086
Custom Editor Variables.....	1090
Configure Toolbars.....	1090
Custom System Properties.....	1092
Localizing the User Interface.....	1094
Setting a Java Virtual Machine Parameter in the Launcher Configuration File / Start-up Script.....	1095
Setting Parameters for the Application Launchers.....	1095
Setting Parameters in the Command Line Scripts.....	1096

Chapter 22: Common Problems.....1097

Performance Problems.....	1098
Large Documents.....	1098
External Processes.....	1098
Display Problems on Linux or Solaris.....	1098
Common Problems and Solutions.....	1098
XML Document Takes a Long Time to Open.....	1098
Oxygen XML Editor Takes Several Minutes to Start on Mac.....	1099
Out Of Memory Error When I Open Large Documents.....	1099
Special Characters Are Replaced With a Square in Editor.....	1099
XSLT Debugger Is Very Slow.....	1099
The Scroll Function of my Notebook's Trackpad is Not Working.....	1099
NullPointerException at Startup on Windows XP.....	1100
Crash at Startup on Windows with an Error Message About a File nvoglv32.dll.....	1100
Oxygen XML Editor Crashed on My Mac OS X Computer.....	1100
Wrong Highlights of Matched Words in a Search in User Manual.....	1100
Keyboard Shortcuts Do Not Work.....	1101
After Installing Oxygen XML Editor I Cannot Open XML Files in Internet Explorer Anymore.....	1101
I Cannot Associate Oxygen XML Editor With a File Type on My Windows Computer.....	1101
The Files Are Opened in Split Panels When I Restart Oxygen XML Editor.....	1101
Grey Window on Linux With the Compiz / Beryl Window Manager.....	1102
Drag and Drop Without Initial Selection Does Not Work.....	1102
Set Specific JVM Version on Mac OS X.....	1102
Segmentation Fault Error on Mac OS X.....	1102
Damaged File Associations on OS X.....	1102
I Cannot Connect to SVN Repository From Repositories View.....	1103
Problem Report Submitted on the Technical Support Form.....	1103
Signature verification failed error on open or edit a resource from Documentum.....	1103
Cannot Cancel a System Shutdown.....	1104
Compatibility Issue Between Java and Certain Graphics Card Drivers.....	1104
An Image Appears Stretched Out in the PDF Output.....	1104
The DITA PDF Transformation Fails.....	1105
The <i>DITA to CHM</i> Transformation Fails.....	1105
DITA Map ANT Transformation Because it Cannot Connect to External Location.....	1106
Topic References outside the main DITA Map folder.....	1106

The PDF Processing Fails to Use the DITA OT and Apache FOP.....	1106
The <i>TocJS</i> Transformation Doesn't Generate All Files for a Tree-Like TOC.....	1107
Navigation to the web page was canceled when viewing CHM on a Network Drive.....	1108
Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x.....	1108
JPEG CMYK Color Space Issues.....	1108
SVG Rendering Issues.....	1108
MSXML 4.0 Transformation Issues.....	1108

Chapter 23: Using the Oxygen XML WebApp.....1109

Oxygen XML WebApp Overview.....	1110
Editing Actions.....	1110
Browser Compatibility.....	1113
License Issues.....	1114

Chapter 24: Customizing Oxygen XML WebApp.....1115

Customization Overview.....	1116
Customizing Oxygen XML WebApp Options.....	1117
Customizing Oxygen XML WebApp Documentation Frameworks.....	1118
Customizing Oxygen XML WebApp Plugins.....	1120
Customizing Oxygen XML WebApp's Client Side.....	1120
Deploying Oxygen XML WebApp.....	1121
Licensing the Oxygen XML WebApp.....	1121
Oxygen XML WebApp How To.....	1122
How To Share a Tomcat Instance Between Oxygen XML WebApp And Another Application.....	1122
How To Make WebApp Use the CMS Authentication Mechanism.....	1122
How To Configure WebApp Minimal File Access Permissions.....	1123
How To Use the WebApp With an WebDAV Server.....	1124

Chapter 25: Comparison Between oXygen XML Author Component and Oxygen XML WebApp.....1125

Glossary.....	1127
---------------	------

Chapter

1

Introduction

Topics:

- *Key Features and Benefits of Oxygen XML Editor*

Welcome to the User Manual of Oxygen XML Editor 17.0!

Oxygen XML Editor is a cross-platform application designed for document development using structured mark-up languages such as XML, XSD, Relax NG, XSL, DTD.

It offers developers and authors a powerful Integrated Development Environment. Based on proven Java technology, the intuitive Graphical User Interface of Oxygen XML Editor is easy to use and provides robust functionality for content editing, project management, and validation of structured mark-up sources. Coupled with *XSLT* and *FOP* transformation technologies, Oxygen XML Editor offers support to generate output to multiple target formats, including: *PDF*, *PS*, *TXT*, *HTML*, *JavaHelp* and *XML*.

Oxygen XML Editor is the XML Editor of choice for developers, authors, and integrators that demand high-quality output with a flexible and robust, single-source, structured mark-up environment.

This user guide is focused mainly at describing features, functionality and application interface to help you get started in no time. It also describes the basic process of authoring, management, validation of structured mark-up documents and their transformation to multiple target outputs. It is assumed that you are familiar with the use of your operating system and the concepts related to structured mark-up.

Key Features and Benefits of Oxygen XML Editor

Multiplatform availability: Windows, OS X, Linux, Solaris	Multilanguage support: English, German, French, Italian and Japanese
Visual WYSIWYG XML editing mode based on W3C CSS stylesheets.	Visual DITA Map editor
Closely integrate with the DITA Open Toolkit for generating DITA output	Support for latest versions of document frameworks: DocBook and TEI.
Can be used as standalone desktop application, run through Java Web Start or as an Eclipse plugin	Non blocking operations, you can perform validation and transformation operations in background
Support for XML, XML Schema 1.0 and 1.1, Relax NG , Schematron, DTD, NVDL schemas, XSLT, XSL:FO, WSDL, XQuery, HTML, CSS	Support for XML, CSS, XSLT, XSL-FO.
Validate XML Schema schemas, Relax NG schemas, DTD's, Schematron schemas, NVDL schemas, WSDL, XQuery, HTML and CSS	Manual and automatic validation of XML documents against XML Schema schemas, Relax NG schemas, DTD's, Schematron, and NVDL schemas
Multiple built-in validation engines (Xerces, libxml, Saxon SA, MSXML 4.0, MSXML.NET) and support for custom validation engines (XSV, SQC).	Multiple built-in XSLT transformers (Saxon 6.5, Saxon 9 Enterprise (schema aware), Xalan, libxslt, MSXML 3.0 / 4.0, Microsoft .NET 1.0, Microsoft .NET 2.0), support for custom JAXP transformers.
Support for latest versions of document frameworks: DocBook and TEI.	Compare and merge files and directories
Ready to use FOP support to generate PDF or PS documents	XInclude support
Support for editing remote files over FTP, SFTP, HTTP / WebDAV and HTTPS / WebDAV	Easy error tracking - locate the error source by clicking on it
Visual schema editor with full and logical model views	Generate HTML documentation from XML Schemas
New XML document wizards to easily create documents specifying a schema or a DTD	Context sensitive content assistant driven by XML Schema, Relax NG, DTD, NVDL or by the edited document structure enhanced with schema annotation presenter
XML Catalog support	Unicode support
Conversions from DTD, Relax NG schema or a set of documents to XML Schema, DTD or Relax NG schema	Syntax coloring for XML, DTD, Relax NG compact syntax, Java, C++, C, PHP, Perl, etc
Pretty-printing of XML files	Easy configuration for external FO Processors
Apply XSLT and FOP transformations	XPath search and evaluation support
Preview transformation results as XHTML or XML or in your browser	Support for document templates to easily create and share documents
Import data from a database, Excel, HTML or text file	Convert database structure to XML Schema
Canonicalize and sign documents	XML project manager
Batch validate selected files in project	Fully-fledged client for the Apache Subversion™ (SVN) versioning system with support for SVN 1.7 and SVN 1.8.
Generate large sets of sample XML instances from XML Schema	Tree view / edit support for XML documents

Configurable external tools	Configurable actions key bindings
Multi-line find and replace support allows regular expressions, is XML aware, is incremental, handles multiple files	Special viewer for very large files (up to 2 GB file size).
Associate extensions on Windows	Bookmark support
OS X ready	Print documents
XSLT Debugger with Backmapping support	XSLT Profiler
XQuery Debugger with Backmapping support	XQuery Profiler
Model View	Attributes View
Multi-document environment	SVG Viewer
XQuery 1.0 and XQuery 3.0 support	WSDL analysis and SOAP requests support
XSLT 2.0 and XSLT 3.0 full support	XPath 2.0 and XPath 3.0 execution and debugging support
Dockable views and editors	Document folding
XSLT refactoring actions	Text transparency levels adjuster
Spell checking supporting English, German and French including locals	Custom protocol plugin support
All the usual editor capabilities (cut, copy, paste, find, replace, windows management)	Drag&drop support
Support for editing, modifying and using files directly from ZIP-type archives	Outline view in sync with a non well-formed document

Chapter

2

Installation

Topics:

- [*Installation Options for Oxygen XML Editor*](#)
- [*Install Oxygen XML Editor on Windows*](#)
- [*Install Oxygen XML Editor on Mac OS X*](#)
- [*Install Oxygen XML Editor on Linux*](#)
- [*Installing Oxygen XML Editor on Windows Server*](#)
- [*Installing Oxygen XML Editor on a Linux / UNIX Server*](#)
- [*Installing Oxygen XML Editor using the Java Web Start \(JWS\) Installer*](#)
- [*Site-wide Deployment*](#)
- [*Obtaining and Registering a License Key for Oxygen XML Editor*](#)
- [*Setting Up a Floating License Server for Oxygen XML Editor*](#)
- [*Transferring or Releasing a License Key*](#)
- [*Upgrading Oxygen XML Editor*](#)
- [*Installing and Updating Add-ons in Oxygen XML Editor*](#)
- [*Uninstalling Oxygen XML Editor*](#)
- [*Oxygen XML Editor Installer Command Line Reference*](#)

The platform requirements and installation instructions are presented in this chapter.

Installation Options for Oxygen XML Editor

Choosing how Oxygen XML Editor runs

You can install Oxygen XML Editor to run in a number of ways:

- As a desktop application on [Windows](#), [Linux](#), or [Mac](#).
- As a desktop application on a [Unix or Linux server](#) or on [Windows Terminal Server](#).
- From within a browser though the [Java Web Start](#) technology.

Choosing an installer

You have a choice of installers;

- The native installer for your platform. On Windows and Linux, the native installer can run also in unattended mode.
- The All-platforms installer, which can be used on any supported platform.

The installation packages were checked before publication with an antivirus program to make sure they are not infected with viruses, trojan horses, or other malicious software.

Choosing a license option

You must [obtain and register a license](#) key to run Oxygen XML Editor.

You can choose from two kinds of license:

- A named-person license, which can be used by a single person on multiple computers.
- A floating license, which can be used by different people at different times. Only one person can use a floating license at a time.

Upgrading, transferring, and uninstalling.

You can also [upgrade](#) Oxygen XML Editor, [transfer a license](#), or [uninstall](#) Oxygen XML Editor.

Getting help with installation

If you need help at any point during these procedures, please send us an email at support@oxygenxml.com.

Install Oxygen XML Editor on Windows

Choosing an installer

You can install Oxygen XML Editor on Windows using one of the following methods:

- Install using the [Windows installer](#).
- Install using the [Windows installer in unattended mode](#).
- Install using the [All Platforms installer](#). Choose the all platforms installer if you have trouble installing using the Windows installer.

System Requirements

System requirements for a Windows install:

Operating systems

Windows Vista, Windows 7, Windows 8, Windows Server 2008, Windows Server 2012

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 GHz
- Recommended - Dual Core class processor

Memory

- Minimum - 2 GB of RAM
- Recommended - 4 GB of RAM

Storage

- Minimum - 400 MB free disk space
- Recommended - 1 GB free disk space

Java

Oxygen XML Editor requires Java. If you use the native Windows installer, Oxygen XML Editor will be installed with its own copy of Java. If you use the all platforms installer, your system must have a compatible Java virtual machine installed.

Oxygen XML Editor supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.7) from Oracle available at

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Oxygen XML Editor may work with JVM implementations from other vendors, but there is no guarantee that those implementations will work with future Oxygen XML Editor updates and releases.

Oxygen XML Editor uses the following rules to determine which installed version of Java to use:

1. If you install using the native Windows installer, which installs a version of Java as part of the Oxygen XML Editor installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Windows environment variable `JAVA_HOME` is set, Oxygen XML Editor uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your `PATH` environment variable is used.

If you run Oxygen XML Editor using the batch file, `oxygen.bat`, you can edit the batch file to specify a particular version to use.

Install using the Windows installer

To install Oxygen XML Editor using the Windows installer:

1. Make sure that your system meets the [system requirements](#).
2. Download the Windows installer.
3. Validate the integrity of the downloaded file by [checking it against the MD5 sum](#) published on the download page.
4. Run the installer and follow the instructions in the installation program.
5. Start Oxygen XML Editor using one of the following methods:
 - Using one of the shortcuts created by the installer.
 - By running `oxygen.bat`, which is located in the install folder.
6. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your [license information](#).

Unattended Installation

You can run the installation in unattended mode by running the installer from the command line with the `-q` parameter. By default, running the installer in unattended mode installs Oxygen XML Editor with the default options and does not overwrite existing files. You can change many options for the unattended installer using the [installer command line parameters](#).

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygen.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
Oxygen XML Editor is now installed in a new sub-folder called `oxygen`.
3. If you wish, you can move the directory where you installed Oxygen XML Editor to your applications directory.
You can also rename it to contain the product version information. For example you can rename it as `oxygen17.0`.
4. Start Oxygen XML Editor by running `oxygen.bat`, which is located in the install directory.
5. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your *license information*.

Install Oxygen XML Editor on Mac OS X

Choosing an installer

You can install Oxygen XML Editor on Mac OS X using one of the following methods:

- Install using the Mac OS X installation package, `oxygen.zip`.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Mac OS X archive installation.

System Requirements

System requirements for a Mac OS X install:

Operating system

Mac OS X version 10.5 64-bit or later

CPU

- Minimum - Intel-based Mac, 1 GHz
- Recommended - Dual Core class processor

Memory

- Minimum - 2 GB of RAM
- Recommended - 4 GB of RAM

Storage

- Minimum - 400 MB free disk space
- Recommended - 1 GB free disk space

Java

Oxygen XML Editor requires Java to run. OS X includes Java by default or it will install it on the first attempt to run a Java application.

Oxygen XML Editor supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0 from Apple). Oxygen XML Editor may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Oxygen XML Editor updates and releases.

Oxygen XML Editor uses the following rules to determine which installed version of Java to use:

1. If you start oXygen with the application launcher (.app) file then:
 - a. if you use the zip distribution for OS X Oxygen XML Editor uses the Apple Java SE 6 available on your Mac computer

- b. if you use the tar.gz distribution that contains a bundled JRE then Oxygen XML Editor will use that bundled JRE
2. If you start Oxygen XML Editor using a startup .sh script then:
 - a. if a bundled JRE is available then it will be used
 - b. otherwise, if the JAVA_HOME environment variable is set then the Java distribution indicated by it will be used
 - c. otherwise the version of Java pointed to by your PATH environment variable will be used

If you run Oxygen XML Editor using the oxygen.sh script, you can change the version of Java used by editing to script file. Go to the Java command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java "-Xdock:name= ..."
```

OS X Installation

To install Oxygen XML Editor on OS X:

1. Download the OS X installation package (oxygen.zip).

The Safari web browser should recognize and expand the compressed file. If it is not automatically expanded, you can expand it manually by double-clicking it.

2. Validate the integrity of the downloaded file by [checking it against the MD5 sum](#) published on the download page.
3. In **Finder**, move the expanded folder to your Applications folder.
Oxygen XML Editor is now installed.
4. Start Oxygen XML Editor, using one of the following methods:
 - Double click Oxygen XML Editor.app.
 - Run sh oxygenMac.sh on the command line.

 **Notice:** You can start more than one instance on the same computer by running the following command for each new instance:

```
open -n Oxygen.app
```

5. To license your copy of Oxygen XML Editor, go to **Help > Register...** to enter your [license key](#).

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (oxygen.tar.gz) to a folder of your choice.
2. Extract the archive in that folder.
Oxygen XML Editor is now installed in a new sub-folder called oxygen.
3. If you wish, you can move the directory where you installed Oxygen XML Editor to your applications directory.
You can also rename it to contain the product version information. For example you can rename it as oxygen17.0.
4. Start Oxygen XML Editor by running oxygenMac.sh, which is located in the install folder.
5. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your [license information](#).

Install Oxygen XML Editor on Linux

Choosing an installer

You can install Oxygen XML Editor on Linux using any of the following methods:

- Install using the Linux installer.
- Install using the Linux installer in unattended mode.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Linux installer.

System Requirements

System requirements for a Linux install:

Operating system

Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.6.0 or later from Oracle

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 GHz
- Recommended - Dual Core class processor

Memory

- Minimum - 2 GB of RAM
- Recommended - 4 GB of RAM

Storage

- Minimum - 400 MB free disk space
- Recommended - 1 GB free disk space

Java

Oxygen XML Editor requires Java. Oxygen XML Editor supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0) from Oracle available at <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Oxygen XML Editor may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Oxygen XML Editor updates and releases. Oxygen XML Editor does not work with the GNU libgcj Java Virtual Machine.

Oxygen XML Editor uses the following rules to determine which installed version of Java to use:

1. If you used the Linux installer, which installs a version of Java as part of the Oxygen XML Editor installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Linux environment variable `JAVA_HOME` is set, Oxygen XML Editor uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your `PATH` environment variable is used.

You can also change the version of the Java Virtual Machine that runs Oxygen XML Author by editing the script file, `oxygen.sh`. Go to the Java command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:

```
/usr/bin/jre1.6.0_45/bin/java -Xmx256m ...
```

Linux Installation

Linux installation procedure.

To install Oxygen XML Editor on Linux:

1. Download the Linux installer.
2. Validate the integrity of the downloaded file by [checking it against the MD5 sum](#) published on the download page.
3. Run the installer that you downloaded and follow the instructions presented in the installation program.
4. Start Oxygen XML Editor using one of the following methods:
 - Use the `oxygen` shortcut created by the installer.
 - Run `sh oxygen.sh` from the command line. This file is located in the installation folder.

- To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your *license key*.

Unattended Installation

You can run the installation in unattended mode by running the installer from the command line with the -q parameter. By default, running the installer in unattended mode installs Oxygen XML Editor with the default options and does not overwrite existing files. You can change many options for the unattended installer using the *installer command line parameters*.

Install using the all platforms installer

To install using the all platforms installer:

- Download the all platforms installation package (`oxygen.tar.gz`) to a folder of your choice.
- Extract the archive in that folder.
Oxygen XML Editor is now installed in a new sub-folder called `oxygen`.
- If you wish, you can move the directory where you installed Oxygen XML Editor to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `oxygen17.0`.
- Start Oxygen XML Editor by running `oxygen.sh`, which is located in the install folder.
- To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your *license information*.

Installing Oxygen XML Editor on Windows Server

Choosing an installer

You can install Oxygen XML Editor on Windows using one of the following methods:

- Install using the *Windows installer*.
- Install using the *Windows installer in unattended mode*.
- Install using the *All Platforms installer*. Choose the all platforms installer if you have trouble installing using the Windows installer.

System Requirements

System requirements for a Windows Server install:

Operating systems

Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 GHz
- Recommended - Dual Core class processor

Memory

- Minimum values per user - 512 MB of RAM
- Recommended values per user - 2 GB of RAM

Storage

- Minimum - 400 MB free disk space
- Recommended - 1 GB free disk space

Java

Oxygen XML Editor requires Java. If you use the native Windows installer, Oxygen XML Editor will be installed with its own copy of Java. If you use the all platforms installer, your system must have a compatible Java virtual machine installed.

Oxygen XML Editor supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.7) from Oracle available at <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Oxygen XML Editor may work with JVM implementations from other vendors, but there is no guarantee that those implementations will work with future Oxygen XML Editor updates and releases.

Oxygen XML Editor uses the following rules to determine which installed version of Java to use:

1. If you install using the native Windows installer, which installs a version of Java as part of the Oxygen XML Editor installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Windows environment variable `JAVA_HOME` is set, Oxygen XML Editor uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your `PATH` environment variable is used.

If you run Oxygen XML Editor using the batch file, `oxygen.bat`, you can edit the batch file to specify a particular version to use.

Install using the Windows installer

To install Oxygen XML Editor using the Windows installer:

1. Make sure that your system meets the [system requirements](#).
2. Download the Windows installer.
3. Validate the integrity of the downloaded file by [checking it against the MD5 sum](#) published on the download page.
4. Run the installer and follow the instructions in the installation program.
5. Start Oxygen XML Editor using one of the following methods:
 - Using one of the shortcuts created by the installer.
 - By running `oxygen.bat`, which is located in the install folder.
6. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your [license information](#).

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygen.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
Oxygen XML Editor is now installed in a new sub-folder called `oxygen`.
3. If you wish, you can move the directory where you installed Oxygen XML Editor to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `oxygen17.0`.
4. Start Oxygen XML Editor by running `oxygen.bat`, which is located in the install directory.
5. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your [license information](#).

Configuring Windows Terminal Server

Windows Terminal Server configuration procedure.

1. Install Oxygen XML Editor on the server and make its shortcuts available to all users.
2. If you need to run multiple instances of Oxygen XML Editor, make sure you add the `-Dcom.oxygenxml.MultipleInstances=true` parameter in the `.bat` startup script.
3. Make sure you allocate sufficient memory to Oxygen XML Editor by adding the `-Xmx` parameter either in the `.bat` startup script, or in the `.vmoptions` configuration file (if you start it from an executable launcher).

Installing Oxygen XML Editor on a Linux / UNIX Server

Choosing an installer

You can install Oxygen XML Editor on Linux using any of the following methods:

- Install using the Linux installer.
- Install using the Linux installer in unattended mode.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Linux installer.

System Requirements

System requirements for a Linux install:

Operating system

Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.6.0 or later from Oracle

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 GHz
- Recommended - Dual Core class processor

Memory

- Minimum - 2 GB of RAM
- Recommended - 4 GB of RAM

Storage

- Minimum - 400 MB free disk space
- Recommended - 1 GB free disk space

Java

Oxygen XML Editor requires Java. Oxygen XML Editor supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0) from Oracle available at <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Oxygen XML Editor may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Oxygen XML Editor updates and releases. Oxygen XML Editor does not work with the GNU libgcj Java Virtual Machine.

Oxygen XML Editor uses the following rules to determine which installed version of Java to use:

1. If you used the Linux installer, which installs a version of Java as part of the Oxygen XML Editor installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Linux environment variable `JAVA_HOME` is set, Oxygen XML Editor uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your `PATH` environment variable is used.

You can also change the version of the Java Virtual Machine that runs Oxygen XML Author by editing the script file, `oxygen.sh`. Go to the `Java` command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:

```
/usr/bin/jre1.6.0_45/bin/java -Xmx256m ...
```

Linux Installation

Linux installation procedure.

To install Oxygen XML Editor on Linux:

1. Download the Linux installer.
2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
3. Run the installer that you downloaded and follow the instructions presented in the installation program.
4. Start Oxygen XML Editor using one of the following methods:
 - Use the `oxygen` shortcut created by the installer.
 - Run `sh oxygen.sh` from the command line. This file is located in the installation folder.
5. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your *license key*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygen.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
Oxygen XML Editor is now installed in a new sub-folder called `oxygen`.
3. If you wish, you can move the directory where you installed Oxygen XML Editor to your applications directory.
You can also rename it to contain the product version information. For example you can rename it as `oxygen17.0`.
4. Start Oxygen XML Editor by running `oxygen.sh`, which is located in the install folder.
5. To license your copy of Oxygen XML Editor go to **Help > Register...** and enter your *license information*.

Unix / Linux Server Configuration

To install Oxygen XML Editor on a Unix / Linux server:

1. Install Oxygen XML Editor on the server and make sure the `oxygen.sh` script is executable and the installation directory is in the PATH of the users that need to use the application.
2. If you need to run multiple instances of the Oxygen XML Editor, make sure you add the `-Dcom.oxygenxml.MultipleInstances=true` parameter in the startup script.
3. Make sure you allocate sufficient memory to Oxygen XML Editor by setting an appropriate value for the `-Xmx` parameter in the `.sh` startup script.
4. Make sure the X server processes located on the workstations allow connections from the server host. For this, use the `xhost` command.
5. Start telnet (or ssh) on the server host.
6. Start an xterm process, with the `display` parameter set on the current workstation. For example: `xterm -display workstationip:0.0`.
7. Start Oxygen XML Editor by typing `oxygen.sh`.

Installing Oxygen XML Editor using the Java Web Start (JWS) Installer

Oxygen XML Editor provides the tools to create your own JWS distribution that can be installed on a custom web server. Advantages of a JWS distribution include:

- Oxygen XML Editor is run locally, not inside a web browser, overcoming many of the browser compatibility problems common to applets.
- JWS ensures that the most current version of the application will be deployed, as well as the correct version of JRE.
- Applications launched with Java Web Start are cached locally. Thus, an already downloaded application is launched on par with a traditionally installed application.
- You can preconfigure Oxygen XML Editor and the rest of your team will use the same preferences and frameworks.



Note: If you want to create your own JWS distribution package, please contact Syncro Soft for the Oxygen SDK agreement.



Note: A code signing certificate is needed to sign the JWS distribution. The following procedure assumes that you already have such a certificate (for example Thawte™, or Verisign™).

The following schematics depicts the Oxygen XML Editor Java Web Start deployment procedure:

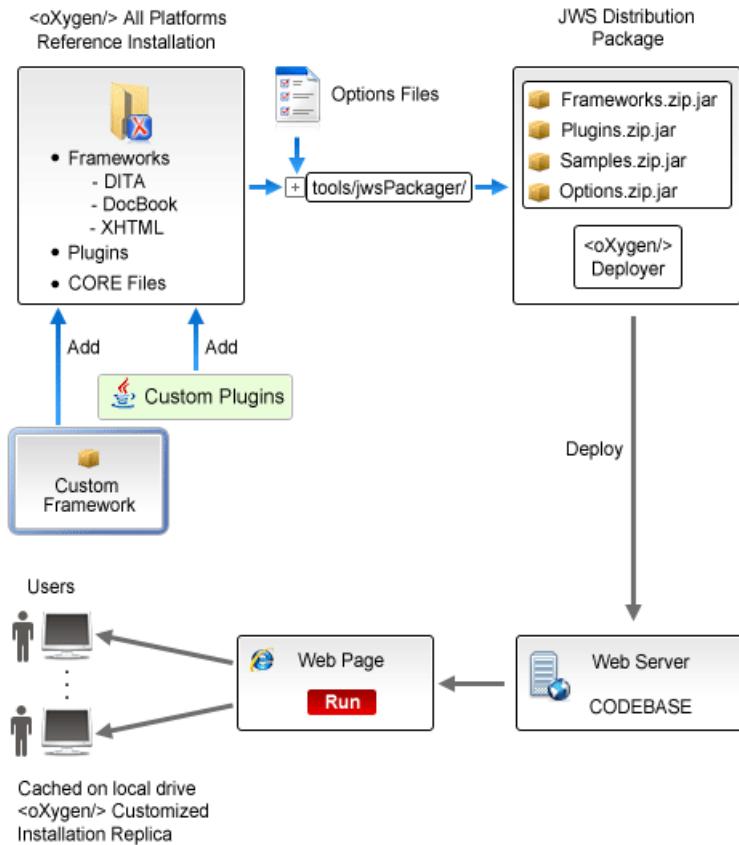


Figure 1: Java Web Start Deployment Procedure

To deploy an Oxygen XML Editor installation on a custom server.

1. Go to <http://www.oxygenxml.com/InstData/Editor/All/oxygen.tar.gz> and download the All Platforms Installation package to a local drive.
2. Expand the archive.
The oxygen folder is created.
3. Optionally, you can customize the content of the **frameworks** folder.
4. Edit the `oxygen\tools\jwsPackager\packager.properties` configuration file. Adjust the following properties appropriately for your server:
 - **codebase** - Represents the location of the future JWS distribution.
 - **keystore** - The *keystore* location path.
 - **storepass** - The password for *keystore* integrity.
 - **storetype** - The type of the certificate file, such as PKCS12 or JKS.
 - **alias** - The *keystore* alias.
 - **optionsDir** - Points to the options directory that may be distributed with the JWS installer. If the directory contains an XML document named `options.xml` or `default.xml` containing exported options, these options will be used. Otherwise, the structure of the options folder has to match the structure of a stand alone application `options folder`.



Note: This property is optional. It is provided only if *custom options* need to be delivered to the end users.

The values of **keystore**, **storepass**, and **alias** properties are all provided by the code signing certificate. For more information, please check the documentation for the *jarsigner* tool.

5. Edit the JNLP `oxygen\tools\jwsPackager\dist\javawebstart\oxygen\oxygen.jnlp` template file to modify default settings. You can specify the list of files opened at startup by modifying the `<argument>` list. To pass system properties directly to Oxygen XML Editor when it is started, add the `oxy` prefix to them (for example: `<property name="oxyPropertyName" value="testValue" />`). The system property is passed to Oxygen XML Editor with the prefix stripped.
6. Open a command-line console and run `ant` in the `oxygen\tools\jwsPackager` folder. The `ant` process creates the `oxygen\tools\jwsPackager\dist\InstData\oxygenJWS.zip` archive that contains the actual remote JWS installer.
7. Copy the expanded content of the archive to the folder specified in the **codebase** property, previously set in the `packager.properties` file.
8.  **Important:** When running the Java Web Start distribution on OS X, due to changes in this [security release](#), clicking the link to the JNLP file does not start the application. The selected JNLP is downloaded locally. Right click it and choose to open the resource.

Using your favorite web browser, go to the address specified in the **codebase** property or to its parent folder and start the remote installer.

Site-wide Deployment

If you are deploying Oxygen XML Editor for a group, there are a number of things you can do to customize Oxygen XML Editor for your users and to make the deployment more efficient.

Creating custom default options

You can [create a custom set of default options](#) for Oxygen XML Editor. These will become the default options for each of your users, replacing Oxygen XML Editor's normal default settings. Users can still set options to suit themselves in their own copies of Oxygen XML Editor, but if they choose to reset their options to defaults, the custom defaults that you set will be used.

Creating default project files

Oxygen XML Editor project files are used to configure a project. You can create and deploy default project files for your projects so that your users will have a preconfigured project file to begin work with.

Shared project files

Rather than each user having their own project file, you can create and deploy shared project files so that all users share the same project configuration and settings and automatically inherit all project changes.

Using the unattended installer

You can speed up the installation process by using the [unattended installer for Windows](#) or [Linux installs](#).

Using floating licenses

If you have a number of people using Oxygen XML Editor on a part-time basis or in different time zones, you can use a [floating license](#) so that multiple people can share a license.

Obtaining and Registering a License Key for Oxygen XML Editor

Oxygen XML Editor is not free software. To enable and use Oxygen XML Editor, you need a license.

For demonstration and evaluation purposes, a time limited license is available upon request at <http://www.oxygenxml.com/register.html>. This license is supplied at no cost for a period of 30 days from the date of issue. During this period, the software is fully functional, enabling you to test all its functionality. To continue using the software after the trial period, you must purchase a permanent license. If a trial period greater than 30 days is required, please contact support@oxygenxml.com.

Choosing a license type

You can use one of the following license types with Oxygen XML Editor:

1. A named-user license may be used by a single named user on one or more computers. Named-user licenses are not transferable to a new named user. If you order multiple named-user licenses, you will receive a single license key good for a specified number of named users. It is your responsibility to keep track of the named users that each license is assigned to.
2. A floating license may be used by any user on any machine. However, the total number of copies of Oxygen XML Editor in use at one time must not be more than the number of floating licenses available. A user who runs two different distributions of Oxygen XML Editor (for example Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single floating license.
3. A special academic license (classroom, department or site license) may be used by students and teachers in academic institutions. These licenses provide a cost effective way of getting access to oXygen for learning purposes.

For definitions and legal details of the license types, consult the End User License Agreement available at <http://www.oxygenxml.com/eula.html>.

Obtaining a license

You can obtain a license for Oxygen XML Editor in one of the following ways:

- You can purchase one or more licenses from the Oxygen XML Editor website at <http://www.oxygenxml.com/buy.html>. A license key will be sent to you by email.
- If your company or organization has purchased licenses please contact your license administrator to obtain a license key.
- If you purchased a subscription and you received a registration code, you can use it to obtain a license key from <http://www.oxygenxml.com/registerCode.html>. A license key will be sent to you by email.
- If you want to evaluate the product you can obtain a trial license key for 30 days from the Oxygen XML Editor website at <http://www.oxygenxml.com/register.html>.

Register a named-user license

To register a named-user license on a machine owned by the named user:

1. Save a backup copy of the message containing the new license key.
2. Start Oxygen XML Editor.

If this is a new install of Oxygen XML Editor, the registration dialog box is displayed. If the registration dialog box is not displayed, go to **Help > Register....**

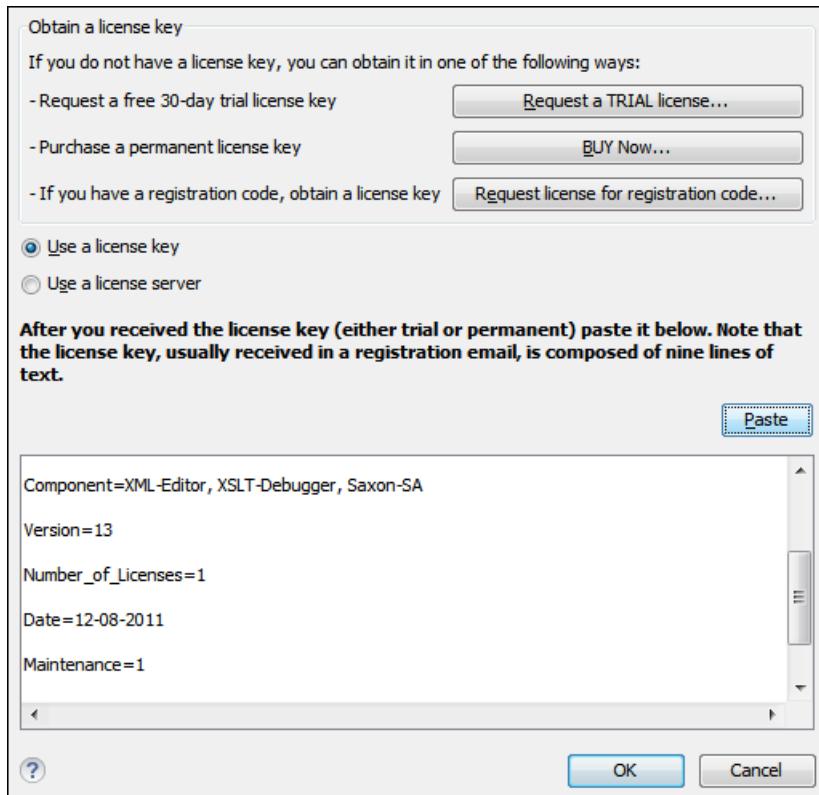


Figure 2: License Registration Dialog Box

3. Select **Use a license key** as licensing method.
4. Paste the license text into the registration dialog box.
5. Press **OK**.

Register Multiple Licenses

If you are installing a named-user license on multiple machines, or you are an administrator registering named-user or floating licenses for multiple users, you can avoid having to open Oxygen XML Editor on each machine by registering the license using a text file or XML file that contains the license information.



Note: If you are using floating licenses that are managed by a license server, you cannot use this method to register licenses.

To register licenses using a text file:

1. Copy the license key to a file named `licensekey.txt` and place it in the installation folder of Oxygen XML Editor

To register licenses using an XML file:

1. Register the license on one computer using the normal *license registration procedure*.
2. Copy the `license.xml` file from the Oxygen XML Editor *preferences directory* on that computer to the installation folder on each installation to be registered.

Registering a floating license

How you register to use a floating license will depend on how floating licenses are managed in your organization.

- If all the machines sharing a pool of floating licenses are on the same network segment, you will register your licence the same way you *register a named-user licence*.



Note: [For System Administrators] Different running instances of Oxygen XML Editor communicate with each other, using UDP broadcast on the 59153 port, to the 239.255.255.255 group.



Warning: This mechanism was deprecated starting with version 17.0 and it is scheduled for removal in a future version. It is recommended to switch to the license server/servlet licensing mechanism.

- If the machines sharing the pool of floating licenses are on different network segments, someone in your company will need to [set up a license server](#). Consult that person to determine if they have set up a license server as a standalone process or as a Java servlet as the registration process is different for each.

Request a Floating License from a License Server Running as a Standalone Process

Use this procedure if your company uses a license server running as a standalone process:

1. Contact your server administrator to get network address and login details for the license server.
2. Start Oxygen XML Editor.
3. Go to **Help > Register**.
The license registration dialog box is displayed.
4. Choose **Use a license server** as licensing method.
5. Select **Standalone server** as server type.
6. In the *Host* field enter the host name or IP address of the license server.
7. In the *Port* field enter the port number used to communicate with the license server.
8. Click the **OK** button.

If a floating license is available, it is registered in Oxygen XML Editor. To display the license details, open the **About** dialog box from the **Help** menu. If a floating license is not available, you will get a message listing the users currently using floating licenses.

Request a Floating License from a License Server Running as a Java Servlet

1. Contact your server administrator to get network address and login details for the license server.
2. Start Oxygen XML Editor.
3. Go to **Help > Register**.
The license registration dialog box is displayed.
4. Choose **Use a license server** as licensing method.
5. Select **HTTP/HTTPS Server** as server type.
6. In the *URL* field enter the address of the license server.
The URL address has the following format:
`http://hostName:port/oxygenLicenseServlet/license-servlet`
7. Complete the *User* and *Password* fields.
8. Click the **OK** button.

If a floating license is available, it is registered in Oxygen XML Editor. To display the license details, open the **About** dialog box from the **Help** menu. If a floating license is not available, you will get a message listing the users currently using floating licenses.

Release a Floating License

The floating license you are using will be released and returned to the pool if:

- The connection with the license server is lost.
- You exit the application running on your machine, and no other copies of Oxygen XML Editor running on your machine are using your floating license.

- You register a named user license with your copy of Oxygen XML Editor, and no other copies of Oxygen XML Editor running on your machine are using your floating license.

Setting Up a Floating License Server for Oxygen XML Editor

Determine if you need to set up a license server

If you are using floating licenses for Oxygen XML Editor, you may need to set up a license server. If the computers that will be using the floating licenses are on different network segments, you must use an Oxygen XML Editor floating license server. A floating license server can be installed as one of the following:

- A *Java servlet*.
- A *standalone process*.

 **Note:** Oxygen XML Editor version 17 or higher requires a license server version 17 or higher. License servers version 17 or higher can be used with any version of a floating license key.

Activating Floating License Keys

To help you comply with the Oxygen XML Editor EULA (terms of licensing), all floating licenses require activation. This means that the license key will be locked to a particular license server deployment and no multiple uses of the same license key are possible.

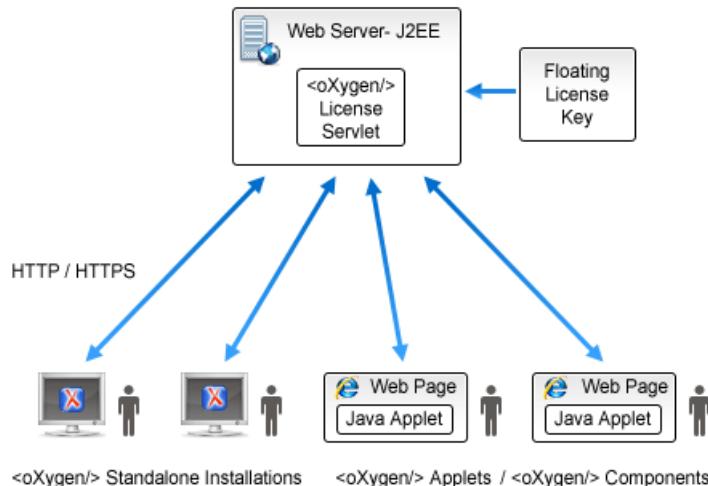
During the activation process, a code that uniquely identifies your license server deployment is sent to the Oxygen XML Editor servers, which in turn will sign the license key.

Split or combine license keys to work with your license servers

A license server can only manage one license key (which can cover any number of floating licenses). If you have multiple license keys for the same Oxygen XML Editor version and you want to have all of them managed by the same server, or if you have a multiple-user floating license and you want to split it between two or more license servers, please contact support@oxygentools.com and ask for a new license key.

Setting up a Floating License Server Running as a Java Servlet

Setting up the floating license server as a servlet.



Steps for Installing the Floating License Server as a Servlet

1. Make sure that Apache Tomcat 5.5 or higher is running on the machine you have selected to be the license server. To get it, go to <http://tomcat.apache.org>.
2. Download the **Web ARchive (.war)** license servlet from the [Oxygen XML Editor website](#).
3. Configure Tomcat to use a security Realm element. Please refer to the [Tomcat Documentation](#) for more information.
4. Edit the `tomcat-users.xml` file from your Tomcat installation and configure one user for each of the following roles: *standard*, *admin*, and *manager*.
5. Go to the Tomcat Web Application Manager page and log-in with the user you configured with the *manager* role. In the **WAR file to deploy** section, choose the WAR file and click the **Deploy** button. The *oXygen License Servlet* is now up and running, but the license keys are not yet registered.
6. Activate the license key. This process involves binding your license key to your license server deployment. Once the process is completed you cannot activate the license with another license server. Follow these steps to activate the license:

- a. Access the license servlet by following the link provided by the Tomcat Web Application Manager page. If prompted for authentication, use the credentials configured for the *admin* or *manager* users.

Result: A page is displayed that prompts for a license key.

- b. Paste your license key into the form and press **Submit**. The browser used in the activation process needs to have Internet access.

Result: You will be redirected to an online form hosted on the Oxygen XML Editor website. This form is pre-filled with an activation code that uniquely identifies your license server deployment, and your license key.

 **Note:** If, for some reason, your browser does not take you to this activation form, refer to the [Manual Activation Procedure](#).

- c. Press **Activate**.

If the activation process is successfully completed, your license server is running. Follow the on-screen instructions to configure the Oxygen XML Editor client applications.

7. By default, the license server logs its activity in the `/usr/local/tomcat/logs/oxygenLicenseServlet.log` file. To change the log file location, edit the `log4j.appenders.R2.File` property from the `/usr/local/tomcat/webapps/oXygenLicenseServlet/WEB-INF/lib/log4j.properties` configuration file.

Manual Activation Procedure

1. Access the license servlet by following the link provided by the Tomcat Web Application Manager page. You will be taken to the license registration page.
2. Copy the license server activation code.
3. Go to the activation page at <http://www.oxygenxml.com/activation/>.
4. Paste the license server activation code and floating license key in the displayed form, then click **Activate**.
5. The activated license key is displayed on-screen. Copy the activated license key and paste it in the license registration page of the servlet.

Report Page

You can access a license server activity report at
<http://hostName:port/oXygenLicenseServlet/license-servlet/report>.

It displays the following real time information:

- **License load** - A graphical indicator that shows how many licenses are available. When the indicator turns red, there are no more licenses available.
- **Floating license server status** - General information about the license server status, including the following information:

- server start time
- license count
- rejected and acknowledged requests
- average usage time
- license refresh and timeout intervals
- location of the license key
- server version
- **License key information** - License key data, including the following information:
 - licensed product
 - registration name
 - company name
 - license category
 - number of floating users
 - Maintenance Pack validity
- **Current license usage** - Lists all currently acknowledged users, including the following information:
 - user name
 - date and time when the license was granted
 - name and IP address of the computer where Oxygen XML Editor runs
 - MAC address of the computer where Oxygen XML Editor runs



Note: The report is also available in XML format at

`http://hostName:port/oxygenLicenseServlet/license-servlet/report-xml.`

Replacing a Floating License Key

The following procedure assumes that your Oxygen XML Editor floating license servlet contains a previously *activated license key*. The following procedure contains instructions for replacing the activated license key with another one. The goal of the procedure is to minimize the license servlet down-time during the activation step of the new license key.

This is useful if, for instance, you want to upgrade your existing license to the latest version or if you receive a new license key *that accommodates a different number of users*.

To replace a floating license key that is activated on your floating license servlet with a new one, follow these steps:

1. Access the license servlet by following the link provided by the Tomcat Web Application Manager page.
 2. Click the **Replace license key** link. This will take you to a page that contains details about the license currently in use.
 3. Click the **Yes** button to begin the replacement procedure.
-
- Note:** During the replacement procedure, new instances of Oxygen XML Editor cannot be licensed by the servlet.
4. Paste the new floating license key in the displayed form, then click **Submit**. The browser used in the process needs to have Internet access.

You will be redirected to an online form hosted on the Oxygen XML Editor website. This form is pre-filled with an activation code that uniquely identifies your license server deployment and your license key.



Note: If, for some reason, your browser does not take you to this activation form, refer to the [Manual Activation Procedure](#).

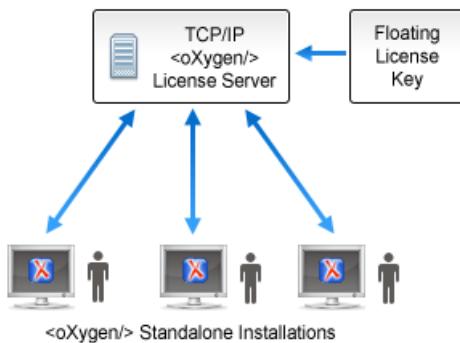
5. Press **Activate**.

If the activation process is successfully completed, your license servlet is running using the new license key. You can click **View license key** to inspect the license key currently used by the license servlet.

If the activation procedure fails, go to step 1 and click **Cancel** to revert to last successfully activated license key.

Setting up a Floating License Server Running as a Standalone Process Using a 32-bit Windows Installer

Setting up the floating license server as a standalone process for Windows.



Steps for Installing the Floating License Server in Windows as a Standalone Process

1. Download the license server installation kit for Windows from the [Oxygen XML Editor website](#).
2. Run the downloaded installer and follow the on-screen instructions.

By default, the installer installs the license server as a Windows service. Optionally, you have the ability to start the Windows service automatically at Windows startup or create shortcuts on the **Start** menu for starting and stopping the Windows service manually. If you want to manually install, start, stop, or uninstall the server as a Windows service, run the following scripts from a command line as an Administrator:

- `installWindowsService.bat [serviceName]` - Installs the server as a Windows service with the name *serviceName*. The parameters for the license key folder and the server port can be set in the `oXygenLicenseServer.vmoptions` file.
- `startWindowsService.bat [serviceName]` - Starts the Windows service.
- `stopWindowsService.bat [serviceName]` - Stops the Windows service.
- `uninstallWindowsService.bat [serviceName]` - Uninstalls the Windows service.



Note: If you do not provide the *serviceName* argument, the default name `oXygenLicenseServer` is used.

If the license server is installed as a Windows service, the output and error messages are automatically redirected to the following log files that are created in the install folder:

- `outLicenseServer.log` - Standard output stream of the server.
- `errLicenseServer.log` - Standard error stream of the server.

3. Manually add the `oXygenLicenseServer.exe` file in the Windows Firewall list of exceptions. Go to **Control Panel > System and Security > Windows Firewall > Allow a program or feature through Windows Firewall > Allow another program** and browse for `oXygenLicenseServer.exe` from the Oxygen License Server installation folder.
4. Floating licenses require activation prior to use. Follow the on-screen instruction to complete the license activation process.



Note: A license server can only manage one license key (which can cover any number of floating licenses). If you have multiple license keys for the same Oxygen XML Editor version and you want to have all of them managed by the same server, or if you have a multiple-user floating license and you want to split it between two or more license servers, please contact support@oxygentools.com and ask for a new license key.

Common Problems

This section includes some common problems that may appear when setting up a floating license server running as a standalone process.

Windows Service Reports "Incorrect Function" When Started

The "Incorrect Function" error message when starting the Windows service usually appears because the Windows service launcher cannot locate a Java virtual machine on your system.

Make sure that you have installed a 32-bit Java SE from Oracle (or Sun) on the system:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

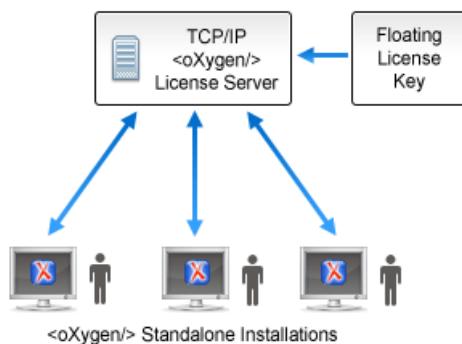
When Started, the Windows Service Reports "Error 1067: The Process Terminated Unexpectedly"

This error message appears if the Windows service launcher quits immediately after being started.

This problem usually happens because the license key has not been correctly deployed (license.txt file in the license folder). For more information, see the [Setting up a Floating License Server section](#).

Setting up a Floating License Server Running as a Standalone Process Using a Platform-independent Distribution

This installation method can be used for running the license server on any platform where a Java virtual machine can run (OS X, Linux/Unix, Windows).



Steps for Installing the Floating License Server as a Standalone Process with a Zip Archive

1. Ensure that a Java runtime version 6 or later is installed on the server machine.
2. Download the license server installation kit for your platform from the [Oxygen XML Editor website](#).
3. Unzip the installation kit into a new folder.
4. Start the server using the startup script from a command line console.

The startup script is called `licenseServer.sh` for OS X and Unix/Linux or `licenseServer.bat` for Windows. The following parameters are accepted:

- `licenseDir` - The path of the directory where the license files will be placed. The default value is `license`.
- `port` - The TCP port number used to communicate with Oxygen XML Editor instances. The default value is `12346`.

The following is an example of the command line for starting the license server on Unix/Linux and OS X:

```
sh licenseServer.sh myLicenseDir 54321
```

5. Floating licenses require activation prior to use. Follow the on-screen instruction to complete the license activation process.

Transferring or Releasing a License Key

If you want to transfer your Oxygen XML Editor license key to another computer (for example if you are disposing of your old computer or transferring it to another person), or release a *floating license* so that someone else can use it, you must first unregister your license. You can then *register your license* on the new computer in the normal way.

1. Go to **Help > Register....**
The license registration dialog box is displayed.
2. The license key field should be empty (this is normal). If it is not empty, delete any text in the field.
3. Make sure the option **Use a license key** is selected.
4. Click **OK**.
A dialog box is displayed asking if you want to reset your license key.
5. Select between falling back to the license key entered previously (for the case of releasing a floating license and reverting to Named User license) and removing your license key from your user account on the computer using the **Reset** button.
The **Reset** button erases all the licensing information. To complete the reset operation, close and restart Oxygen XML Editor.

Upgrading Oxygen XML Editor

From time to time, upgrade and patch versions of Oxygen XML Editor are released to provide enhancements that fix problems, and add new features.

Checking for New Versions of Oxygen XML Editor

Oxygen XML Editor checks for new versions automatically at start up. To disable this check, *open the Preferences dialog box*, go to **Global**, and uncheck **Automatic Version Checking**.

To check for new versions manually, go to **Help > Check for New Versions**.

What is preserved during an upgrade

When you install a new version of Oxygen XML Editor, some data is preserved and some is overwritten. If there is a previous version of Oxygen XML Editor already installed on your computer, it can coexist with the new one, which means you don't have to uninstall it.

If you install over a previously installed version:

- All the files from its install directory will be removed, including any modification in frameworks files, *predefined document type*, XSLT stylesheets, XML catalogs, and templates.
- All global user preferences are preserved and will be imported into the new version.
- All project preferences will be preserved in their project files.
- Any custom frameworks that were stored outside the installation directory (as configured in *Document type associations > Locations*) will be preserved and will be found by the new installation.

If you install in a new directory.

- All the files from the old install directory will be preserved, including any modification in frameworks files, *predefined document type*, XSLT stylesheets, XML catalogs, and templates. However, these modifications will not be automatically imported into the new installation.
- All global user preferences are preserved and will be imported into the new version.
- All project preferences will be preserved in their project files.
- Any custom frameworks that were stored outside the installation directory (as configured in *Document type associations > Locations*) will be preserved and will be found by the new installation.

Upgrading the Standalone Application

1. Upgrading to a new version might require a new license key. To check if your license key is compatible with the new version, select **Help > Check for New Version**. Note that the application needs an Internet connection to check the license compatibility.
2. Download and install the new version according to the instructions for your platform and the type of installer you selected.
3. If you installed from an archive (as opposed to an executable installer) you may have to update any shortcuts you have created or modify the system PATH to point to the new installation folder.
4. Start Oxygen XML Editor.
5. If you require a new license for your upgrade, install it now according to the procedure for your platform and the type of installer you selected.

Installing and Updating Add-ons in Oxygen XML Editor

Oxygen XML Editor provides an add-on mechanism that can automatically discover and install *frameworks* and *plugins* from a remote location.

 **Note:** Frameworks that you install through the add-ons system are read-only.

Installing Add-ons

To install a new add-on, follow these steps:

- Go to **Help > Install new add-ons...**
- In the displayed dialog box, fill-in the **Show add-ons from** with the *update site* that hosts add-ons. The add-ons list contains the name, status, update version, Oxygen XML Editor version, and the type of the add-on (either framework, or plugin). A short description of each add-on is presented under the add-ons list.

 **Note:** To see all the add-ons from the remote update site, disable **Show only compatible add-ons** and **Show only the latest version of the add-ons**. Incompatible add-ons are shown only to acknowledge their presence on the remote update site. You cannot install an incompatible add-on.

- By default, only the latest versions of the add-ons that are compatible with the current version of Oxygen XML Editor are displayed.
- Choose the add-ons you want to install, press the **Next** button, then follow the on-screen instructions.

 **Note:** Accepting the license agreement of the add-on is a mandatory step in the installation process

 **Note:** All add-ons are installed in the *preferences directory* of Oxygen XML Editor, under the *extensions* directory.

Managing installed add-ons

To manage the installed add-ons, follow these steps:

- Go to **Help > Manage add-ons...**
- The displayed dialog box presents a list of the available updates (compatible with the current version of Oxygen XML Editor) and with the already installed updates. Under the updates list, Oxygen XML Editor presents a short description of each update.
- Check for box for a specific add-on, then press **Update** to update it (or **Uninstall** to remove it). If there is a newer version of the add-on available, Oxygen XML Editor will download the package and install it. Follow the on-screen instructions to complete the installation process

 **Note:** Accepting the license agreement of the add-on is a mandatory step in the installation process

Checking for add-on updates

To check if there are available updates for the installed add-ons, go to **Help > Check for add-ons updates....** This action only displays updates that are compatible with the current Oxygen XML Editor version.

To watch a video demonstration about the add-ons support in Oxygen XML Editor, go to <http://www.oxygenxml.com/demo/AddonsSupport.html>.

Uninstalling Oxygen XML Editor

Uninstalling the Oxygen XML Editor Standalone



Caution: The following procedure will remove Oxygen XML Editor from your system. **All data stored in the installation directory will be removed, including any customizations or any other data you have stored within that directory. Please make a back up of any data you want to keep before uninstalling Oxygen XML Editor.**

1. Backup any data you want to keep from the Oxygen XML Editor installation folder.
 2. Remove the application.
 - On Windows use the appropriate uninstaller shortcut provided with your OS.
 - On OS X and Unix manually delete the installation folder and all its contents.
 3. If you want to remove the user preferences:
 - **On Windows XP**, remove the directory: %APPDATA%\com.oxygenxml (usually %APPDATA% has the value [user-home-dir]\Application Data).
 - **On Windows Vista/7/8**, remove the directory: %APPDATA%\Roaming\com.oxygenxml (usually %APPDATA% has the value [user-home-dir]\Application Data). Note that this is directory is hidden.
 - **On Linux**, remove the directory: .com.oxygenxml from the user's home directory.
 - **On Mac OS X**, remove the directory: Library/Preferences/com.oxygenxml of the user home folder

Unattended Uninstall

The unattended uninstall procedure is available only on Windows and Linux.

Run the uninstaller executable from command line with the -q parameter.

The uninstaller executable is called `uninstall.exe` on Windows and `uninstall` on Linux and is located in the application's install folder.

Oxygen XML Editor Installer Command Line Reference

Command line options of the Oxygen XML Editor installer.

The response.varfile

The Oxygen XML Editor installers for Windows and Linux creates a file called `response.varfile`, which records the choices that the user made when running the installer interactively. You can use a `response.varfile` to set the options for an unintended install. Here is an example of a `response.varfile`:

```

sys.programGroupAllUsers$Boolean=true
reportProblem=true
downloadResources=true
sys.languageId=en
sys.installationDir=C:\\Program Files (x86)\\Oxygen XML Editor 16
createQuickLaunchIconAction$Boolean=true
sys.programGroupName=Oxygen XML Editor 16.0
executeLauncherAction$Boolean=true
varfileAdditionalExtensions$StringArray="xml", "dita", "ditaa", "ditaval", "xsl", "xslt", "xsp", "xsd", "xrc", "sch", "dd", "hml", "ext", "hml", "wsdl", "sqery", "xq", "xqy", "xqf", "xqj", "xpl", "css", "json", "xp"

```

The following table describes some of the settings that can be used in the `response.varfile`:

Table 1: response.varfile Options Parameters

Parameter name	Description	Values
<code>autoVersionChecking</code>	Automatic version checking.	true / false. Default setting is true.
<code>reportProblem</code>	Allows you to report a problem encountered while using Oxygen XML Editor.	true / false. Default setting is true.
<code>downloadResources</code>	Allows Oxygen XML Editor to download resources (links to video demonstrations, webinars and upcoming events) from http://www.oxygenxml.com to populate the application welcome screen.	true / false. Default setting is true.

The Oxygen XML Editor installation uses the install4j installer. A description of [the response.varfile format](#) can be found on the install4j site.

Command line parameters

The Oxygen XML Editor installer supports the following command line parameters:

Option	Meaning
<code>-q</code>	Run the installer in unattended mode. The installer will not prompt the user for input during the install. Default settings will be used for all options unless a <code>response.varfile</code> is specified using the <code>-varfile</code> option or individual settings are specified using - on Windows: <pre>oxygen.exe -q</pre> - on Linux: <pre>oxygen.sh -q</pre>
<code>-overwrite</code>	In unattended mode, the installer does not overwrite files with the same name if a previous version of the Oxygen XML Editor is installed in the same folder. The <code>-overwrite</code> parameter added after the <code>-q</code> parameter forces the overwriting of these files. - on Windows: <pre>oxygen.exe -q -overwrite</pre> - on Linux: <pre>oxygen.sh -q -overwrite</pre>
<code>-console</code>	To display a console for the unattended installation, add a <code>-console</code> parameter to the command line. - on Windows: <pre>start /wait oxygen.exe -q -console</pre>

Option	Meaning
	<p> Note: The use of <code>start /wait</code> on Windows is required to make the installer run in the foreground. If you run it without <code>start /wait</code>, it will run in the background.</p> <p>- on Linux:</p> <pre>oxygen.sh -q -console</pre>
<code>-varfile</code>	<p>Points to the location of a <code>response.varfile</code> to be used during an unattended installation. For example:</p> <p>- on Windows:</p> <pre>oxygen.exe -q -varfile response.varfile</pre> <p>- on Linux:</p> <pre>oxygen.sh -q -varfile response.varfile</pre>
<code>-V</code>	<p>Is used to define a variable to be used by an unattended installation. For example:</p> <p>- on Windows:</p> <pre>oxygen.exe -q -VusageDataCollector=false</pre> <p>- on Linux:</p> <pre>oxygen.sh -q -VusageDataCollector=false</pre>

The Oxygen XML Editor installation uses the install4j installer. A full list of the [command line parameters supported by the install4j installer](#) can be found on the install4j site.

Chapter

3

Getting Started

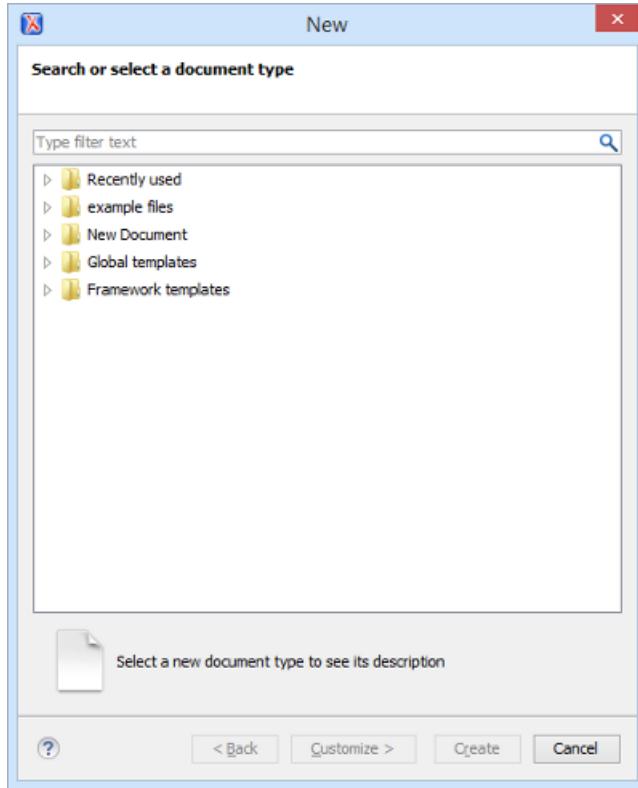
Topics:

- *Your First XML Document*

The basic steps for creating an XML document are presented here.

Your First XML Document

To create your first XML document, select **File > New....** The **New Document Wizard** is displayed:



You can either create a new XML document from scratch by choosing one of the available types in the **New Document** folder, or you can create one from a template by choosing a template from the **Global templates** or **Framework templates** folders. If you are looking for a common document type, such as DITA or DocBook, you can find templates for these document types in the **Framework templates** folder. If your company has created its own templates, you can also find them there. After you use this dialog box to create a few documents, those document types will appear in the **Recently used** folder, which allows you to easily create other new documents of those types.

For some document types, you may find a number of different templates. For example, there are numerous templates for DocBook documents, and DITA topic types and maps. Choose the template that best meets your needs.

Writing Your First Document

Depending on the type of document you choose, the Oxygen XML Editor interface changes to support editing that document type. This may include new menus, toolbar buttons, and items in the contextual menus.

Also, depending on the type of document you choose, Oxygen XML Editor may open your document in **Text** or **Author** mode. **Text** mode shows the raw XML source file, while **Author** mode shows a graphical view of the document.

Whether there is an **Author** mode view available for your document type depends on the type you choose and if there is a CSS stylesheet available to create the **Author** view. Oxygen XML Editor includes default **Author** mode views for most of the document types it supports. If your company has created its own document types, **Author** mode stylesheets may have also been created for that type. However, if you create a plain XML file, or one based on a schema that is not included in the Oxygen XML Editor built-in support, you need to edit it in **Text** mode or [create your own Author mode style sheet](#) for it.

You can switch back and forth between **Author** mode and **Text** mode at any time by clicking the buttons at the bottom left of the editor window. You do not lose any formatting when switching from **Author** to **Text** mode. **Text** and **Author** modes are just different views for the same XML document. There is also a **Grid mode** available, which is useful for

certain kinds of documents, particularly those that are structured like databases. You can also use it to *sort things such as list items and table rows*.

If you use **Author** mode, you might find that it is similar to word processors that you are used to. Likewise, the **Text** mode is similar to many typical text editors. If you are new to XML, the biggest difference is that XML documents have a particular structure that you have to follow. Oxygen XML Editor assists you with a continuous validation of the XML markup.

Structuring Your First Document

Each XML document type has a particular structure that you have to follow as you write and edit the document. Some document types give you a lot of choices, while others give you very few. In either case, you need to make sure that your document follows the particular structure for the document type you are creating. This means:

- At any given location in the document, there are only certain XML elements allowed. Oxygen XML Editor helps you determine which elements are allowed. In **Author** mode, when you press **Enter**, Oxygen XML Editor assumes that you want to enter a new element and shows you a list of elements that can be created in this location. Keep typing until the element you want is highlighted and press **Enter** to insert the element. If you want to view the overall structure of a document and see what is allowed (and where), you can use the **Model** view (**Window > Show View > Model**).
- When you create certain elements, you may find that your text gets a jagged red underline and you get a warning that your content is invalid. This is usually because the element you have just created requires certain other elements inside of it. Your document will be invalid until you create those elements. Oxygen XML Editor does its best to help you with this. If there is only one possible element that can go inside the element you just created, Oxygen XML Editor creates it for you. However, if there is more than one possibility you have to create the appropriate elements yourself.

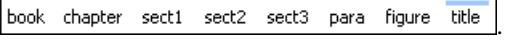
Editing Your First Document

Once you have completed the first draft of your document, you may need to edit it. As with any editor, Oxygen XML Editor provides the normal cut, copy, and paste options as well as drag and drop editing. However, when you are editing an XML document, you have to make sure that your edits respect the structure of the XML document type. In fact, you are often editing the structure as well as the content of your document.

Oxygen XML Editor provides many tools to help you edit your structure and to keep your structure correct while editing text.

The Document Breadcrumbs

Across the top of the editor window, there is a set of breadcrumbs that shows you exactly where the insertion point

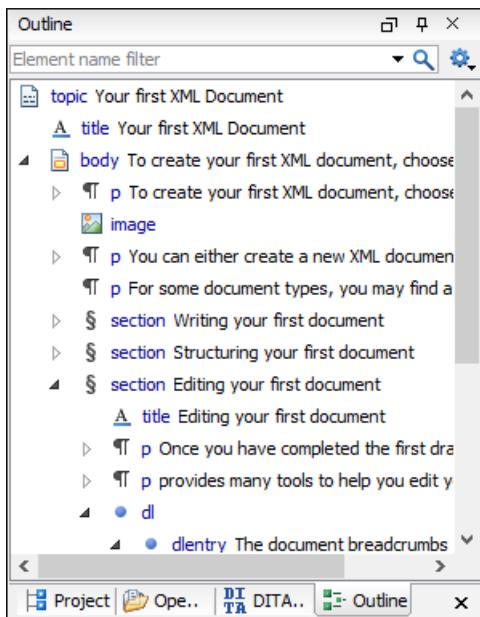
is in the structure of the document.  You can click on any element in the breadcrumbs to select that entire element in the document.

Showing Tags

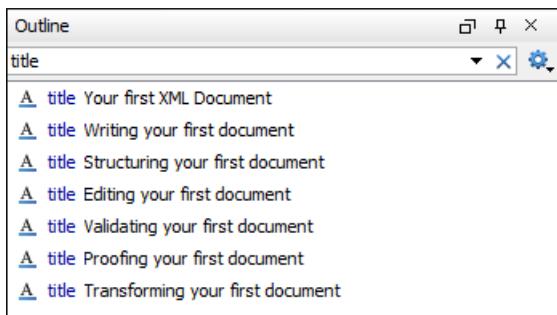
To see exactly where you are in the structure of the document, you can show the tags graphically in the **Author** view. There are several levels of tag visibility that you can choose using the  **Show Tags** drop-down list on the toolbar (the button may look a little different as it changes to reflect the level of tags currently displayed).

Outline View

The **Outline** view shows you the structure of your document in outline format. You can use it to select elements, or to move elements around in the document.



You can configure the **Outline** view to determine what is shown, such as element names, attributes, and comments. Certain choices may work better for particular document types. You can also filter the **Outline** view to show only elements with a certain name.



Cut and Paste, Drag and Drop

You can cut and paste or drag and drop text, just as you would in any other editor. However, when you do this in **Author** view, it is important to remember that you are actually moving blocks of XML. When you cut and paste or drag and drop a block of XML, the result has to be valid both where the content is inserted, and where it is removed from.

A big part of doing this correctly is to make sure that you pick up the right block of text in the first place. Using the breadcrumbs or **Outline** view, or showing tags and using them to select content, can help ensure that you are selecting the right chunk of XML.

If you do try to paste or drop a chunk of XML somewhere that is not valid, Oxygen XML Editor warns you and tries to suggest actions that make it valid (such as by removing surrounding elements from the chunk you are moving, by creating a new element at the destination, or by inserting it in a nearby location).

You can also switch to **Text** view to see exactly which bits of XML you are selecting and moving.

Refactoring actions

You can perform many common structure edits, such as renaming an element or wrapping text in an element, using the actions in the **Document > Markup** menu. The refactoring actions are also available from the contextual menu and on the **Markup** toolbar (right-click the toolbar area to choose which toolbars to show).

Validating Your First Document

Validation is the process of making sure that an XML document abides by the rules of its schema. If Oxygen XML Editor knows how to find the schema, it validates the document for you as you type. (Oxygen XML Editor finds the schema automatically for most of the document types created from templates. However, in some cases you may have to [tell it how to find the schema](#).)

When Oxygen XML Editor validates as you type, there is a small bar at the right edge of the editor that shows you if the document is invalid and where errors are found. If the indicator at the top of that bar is green, your document is valid. If the document is invalid, the indicator turns red and a red flag shows you where the errors are found. Click that flag to jump to the error. Remember that sometimes your document is invalid simply because the structure you are creating is not yet complete.

In addition to problems with the validity of the XML document itself, Oxygen XML Editor also reports warnings for a number of conditions, such as if your document contains a cross reference that cannot be resolved, or if Oxygen XML Editor cannot find the schema specified by the document. The location of these warnings is marked in yellow on the validation bar. If the document contains warnings, but no errors, the validity indicator turns yellow.

You can also validate your document at any time by selecting the  **Validate** action from the  **Validation** toolbar drop-down list or the **Document > Validate** menu.. When you validate in this manner, the validation result opens in a new pane and it shows each validation error on a separate line. Clicking on the error takes you to the location in your document where the error was detected. Be aware that the problem is sometimes in a different location from where the validator detects the error.

To get more information on a validation error, right-click on a validation error message, and select **Show Message**.

Proofing Your First Document

Oxygen XML Editor provides spell checking support. You can check the spelling of your document by pressing the  **Check Spelling...** button or selecting **Edit > Check Spelling...**. Interactive spell checking is also available, although it is disabled by default. You may notice that validation errors are marked the same way spelling errors are marked in ordinary word processors. To turn it on, [open the Preferences dialog box](#) and go to **Editor > Spell Check > Automatic spell check**.

Transforming Your First Document

An XML document must be transformed in order to be published. Transformations are specific to the particular type of document you have created. A DITA transformation cannot be used on a DocBook file, or vice versa. A single document type may have many different transformations that produce different kinds of outputs. For some document types, such as DITA, many different content files may be combined together by a transformation. You need to locate and launch a transformation that is appropriate for your document type and the kind of output you want to generate.

Oxygen XML Editor uses [transformation scenarios](#) to control the transformation process. Depending on the document type you have created, there may be several transformation scenarios already configured for your use. This may include the default transformation scenarios supplied by Oxygen XML Editor or ones created by your organization.

To see the list of transformations available for your document, select the  **Apply Transformation Scenario(s)** action from the toolbar or the **Document > Transformation** menu. A list of available transformation scenarios are displayed. Choose one or more scenarios to apply, and click **Apply associated**. Exactly how your transformed content appears depends on how the transformation scenario is configured.

Chapter

4

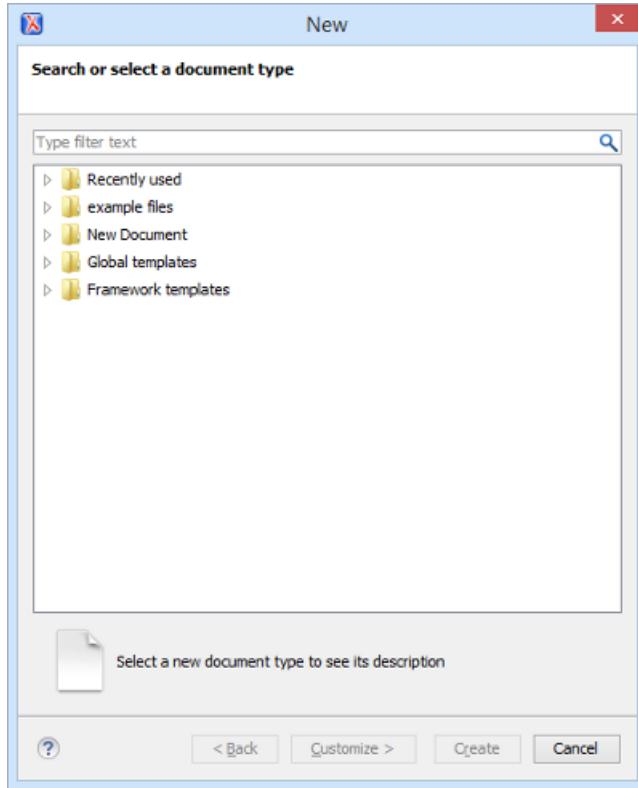
DITA Authoring Guide

Topics:

- *Your First DITA Topic*
- *Topics and Topic Structure*
- *Documents and Maps*
- *Reuse*
- *Linking*
- *Output*
- *Metadata*
- *Background: Keys*

Your First DITA Topic

To create your first DITA topic, choose **File > New...**. The **New Document Wizard** is displayed:



Go to **Framework templates > DITA > topic** and select the type of topic that you want to create.

 **Note:** If your organization has created DITA customizations, the appropriate template files may be in another location, and various different types of topics may be provided for your use. Check with the person who manages your DITA system to see if you should be using templates from a different directory.

Your DITA topic is an XML document, thus all *the editing features that Oxygen XML Editor provides for editing XML documents* also apply to DITA topics. However, Oxygen XML Editor also provides extensive additional support for *editing DITA topics, their associated DITA maps*, and for *creating DITA output*.

Understanding DITA Topics

It is important to understand the role that a DITA topic plays in a DITA project. A DITA topic is not associated with a single published document. It is a separate entity that can potentially be included in many different books, help systems, or websites. Therefore, when you write a DITA topic you are not writing a book, a help system, or a website. You are writing an individual piece of content. This affects how you approach the writing task and how Oxygen XML Editor works to support you as you write.

Most of your topics are actually related to other topics, and those relationships can affect how you write and handle things such as links and content reuse. Oxygen XML Editor helps you manage those relationships. Depending on how your topics are related, you can use the tools provided in Oxygen XML Editor, along with the features of DITA, in a variety of different ways.

The Role of Maps

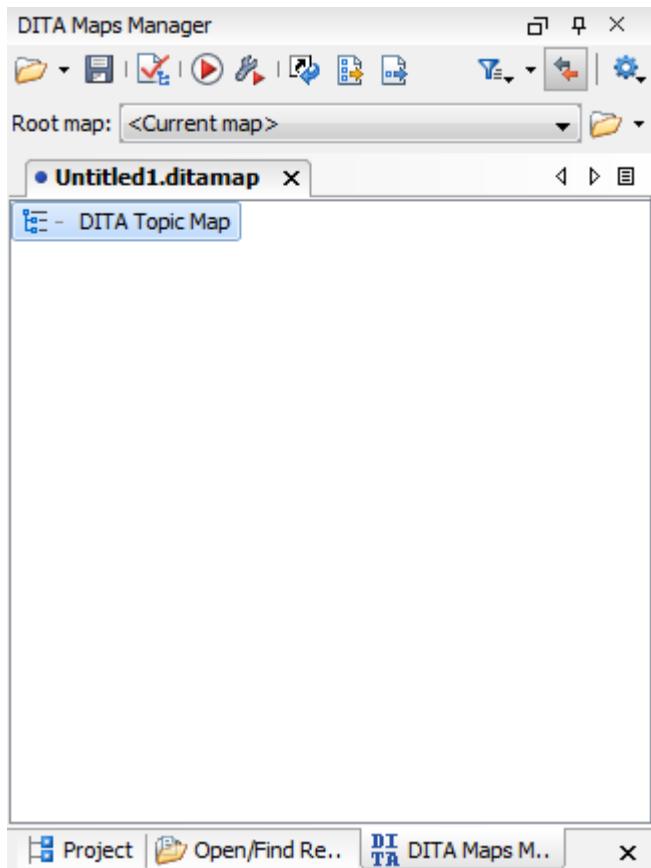
The basic method that DITA uses to express the relationship between topics is through a DITA map. Other relationships between topics, such as cross references, generally need to be made between topics in the same map. DITA uses maps to determine which topics are part of any output that you create. While customized DITA solutions can use other

mechanisms, generally DITA is not used as a way to publish individual topics. Output is created from a map and includes all the topics referenced by the map.

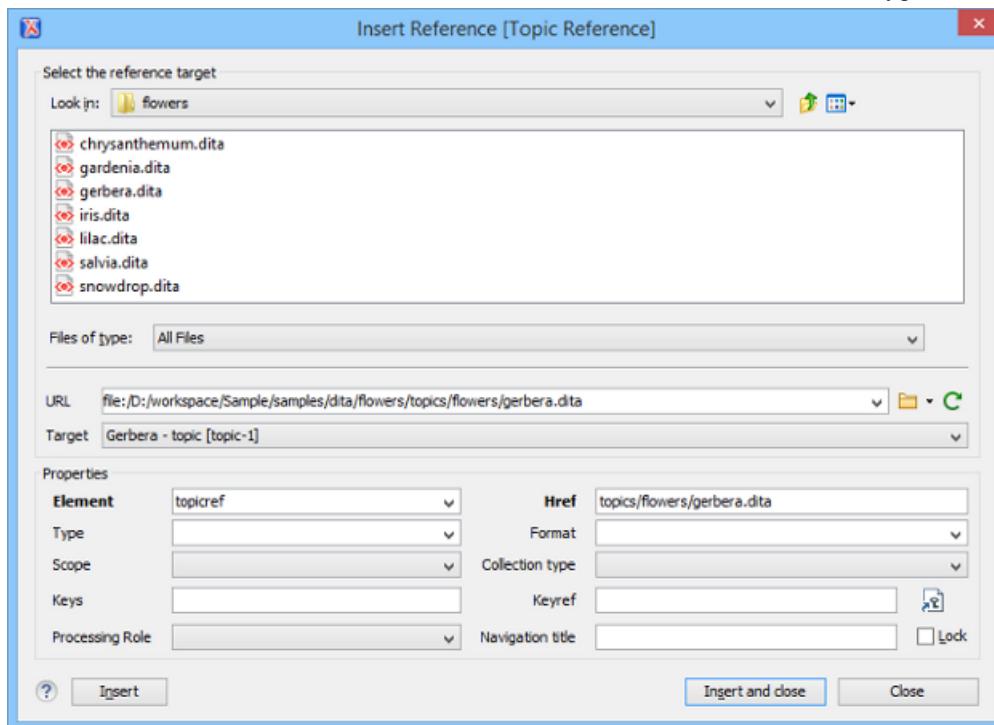
A publication is not always represented by a single map. For instance, if you are writing a book, you might use a map to create each chapter and then organize the chapters in another map to create the book. If you are writing help topics, you might use a map to combine several DITA topics to create a single help topic and then create another map to organize your help topics in a help system. This allows you to reuse the content of a map in multiple projects.

Adding Your Topic to a Map

To add your topic to a map, you must first create the map. A map is an XML document, similar to a topic. To create a map, choose **File > New... > Framework templates > DITA Map > map** and select the type of map (for example, Map or Bookmap). Oxygen XML Editor asks if you want to open your map in the editor or in the **DITA Maps Manager**. Usually, opening it in the **DITA Maps Manager** is the best choice. You can also open the map in the editor from the **DITA Maps Manager**.



The **DITA Maps Manager** presents a view of the DITA map that is similar to a table of contents. To add a topic to a map, add a topic reference to the map using a `topicref` element. The easiest way to do this is to open the topic in the editor, then right-click on **DITA Topic Map** in the **DITA Maps Manager** view and choose **Append child > Reference to the currently edited file...**. This opens the **Insert Reference** dialog box with all of the required fields already filled in for you. You can fill in additional information here or add it to the map later. When you select **Insert and close**, a reference to your topic is added to the map.



If you want to see what the resulting map looks like in XML, save your map and then double-click on **DITA Topic Map** in the **DITA Maps Manager** view. The XML version of the map opens in the editor.

 **Note:** The default title for maps in Oxygen XML Editor is **DITA Topic Map**, which is shown in the **DITA Maps Manager** view. You can change the title by right-clicking on the title of the map and selecting **Edit properties....** For the purposes of this guide, we will continue to use the title **DITA Topic Map**.

Child Topics

As you add topics to your map, you may want to make a topic the child of another topic. Making it a child of another is usually done at the map level. To create a child topic reference, right-click on the parent topic in the **DITA Maps Manager** view and choose **Append child**. You can then choose one of the following options:

- **New topic...** - Opens the **New** file wizard for creating a new topic.
- **Reference...** - Opens the **Insert Reference** dialog box that allows you to create a reference to an existing topic.
- **Reference to the currently edited file...** - Creates a reference to the file that is currently opened in the editor.

You can also change the order and nesting of topics in the **DITA Maps Manager** view by doing either of the following:

- Select the topic to move while holding down the **Alt** key and use the arrow keys to move it around.
- Use the mouse to drag and drop the topic to the desired location.

The way your parent and child topics are organized in any particular output depends on both the configuration of those topics in the map and the rules of the output transformation that is applied to them. Do not assume that your topics must have the same organization for all output types. The map defines the organization of the topics, not the topics themselves. It is possible to create a variety of different maps, each with different organization and configuration options to produce a variety of different outputs.

Child Maps

If you have a large set of information, such as a long book or extensive help system, a single map can become long and difficult to manage. To make it easier to manage, you can break up the content into smaller maps. A smaller map might represent a chapter of a book, a section of a user manual, or a page on a website.

To build a publication out of these smaller maps, you must add them to a map that represents the overall publication. To add a child map to the current map, right-click on the title of the map (for example, **DITA Topic Map**) and choose **Append child > Map reference....**

Validating a Map

Just as it is with your individual topics, it is important to validate your maps. Oxygen XML Editor provides a validation function for DITA maps that does more than simply validating that the XML is correct. It also does the following:

- Validates all of the relationships defined in the maps.
- Validates all of the files that are included in the map.
- Validates all of the links that are expressed in the files.

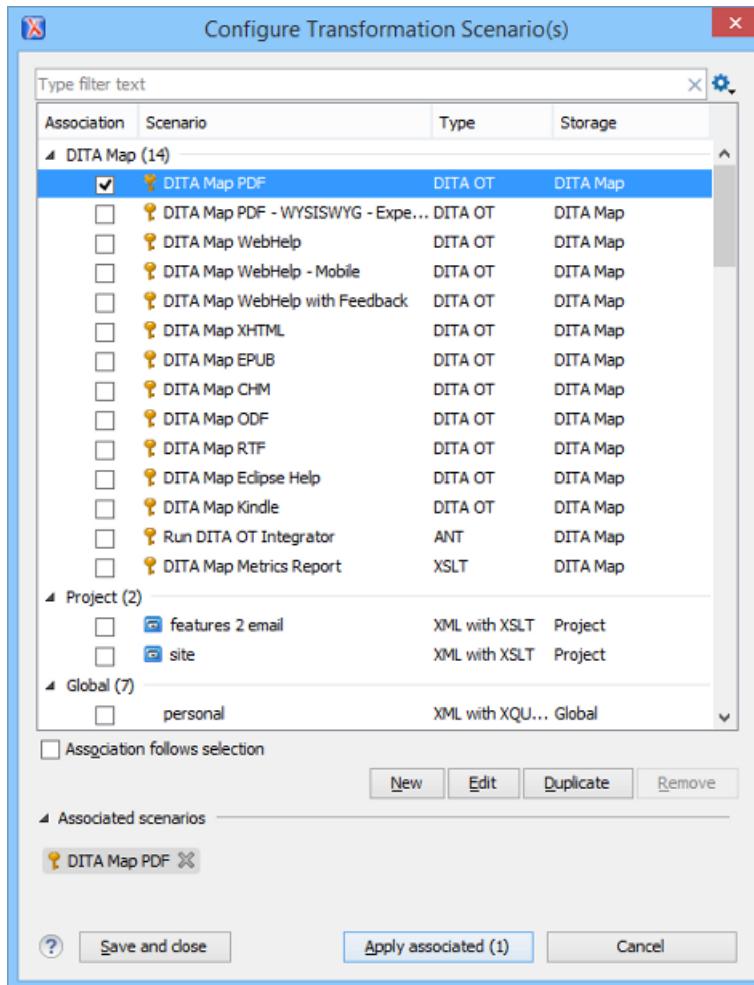
Validating the map that describes your entire publication validates all the files that make up the publication and all of the relationships between them. To validate a map, click the  **Validate and Check for Completeness** button in the **DITA Maps Manager** view.

Publishing Your Topics

As noted previously, in DITA standards you usually do not publish output from an individual topic. Instead, you create published output by running a DITA transformation on a map. This collects all the topics that are referenced in the map, organizes them, and produces output in a particular format. By default, Oxygen XML Editor uses the transformations provided by the **DITA Open Toolkit** for publishing to a variety of different output formats (such as PDF, WebHelp or EPUB). Your organization may have created various custom transformations or modified the built-in **DITA Open Toolkit** transformations. In either case, Oxygen XML Editor manages them by using transformation scenarios.

To publish output for a map, select the transformation scenario you want to run and set any of the parameters it requires.

To select a transformation, click the  **Configure Transformation Scenario(s)...** button in the **DITA Maps Manager** view. This opens the **Configure Transformation Scenario(s)** dialog box.



Choose the transformation scenarios you want to apply and click **Apply associated**. Depending on the configuration of the transformation scenario, when the transformation is finished, your output may automatically be opened in the appropriate application. To change or view the configuration or storage options for a transformation scenario, select the transformation and click **Edit**.

Topics and Topic Structure

DITA is a structured writing format. Structure can have several meanings, all of which are relevant to DITA.

Information Types

The structure of a piece of content refers to how the words and images are selected and organized to convey information. One approach to structured writing is to divide content into discrete blocks containing different types of information, and then to combine those blocks to form publications. DITA is based on this approach, and encourages the author to write in discrete blocks called topics. DITA provides three base topic types (concept, task, and reference), a number of extended topic types, and the capability to create new topic types through specialization.

Text Structure

Every piece of text is made up of certain text structures, such as paragraphs, lists, and tables. DITA supports text structures through XML elements such as p, ol, and simpletable. *The DITA markup* specifies the text structures, but not how they will be published in different types of media. The formatting of text structures is determined by the output transformations and may be customized to meet the needs of a variety of different organizations and media.

Semantic Structure

Semantic structure is structure that shows the meaning of things. For example:

- A `task` element specifies that a block of content contains the description of a task
- A `codeblock` element specifies that a block of text consists of programming code
- A `uicontrol` element specifies that a word is the name of a control in a computer GUI
- The `platform` profiling attribute specifies that a particular piece of content applies only to certain computing platforms

Semantic structure is important in a structured writing system because it allows both authors and readers to find content, and it allows processing scripts to process various pieces of content differently, based on their role or meaning. This can be used to do things such as filtering content related to a specific product so that you can produce documentation on many products from the same source.

There can be many forms of semantics captured in a document set. DITA captures some of these in topics and some of them in maps. If you are using a CMS, it may capture additional semantics.

Document Semantics

Documents consist of a number of different elements that may be made up of the same basic text structures as the rest of the text, but have a special function within the structure of the document. For instance, both tables of contents and indexes are lists, but they play a special role in the document. Chapters and sections are just sequences of paragraphs and other text structures, yet they are meaningful in the structure of the document. In some cases, such as indexes and tables of contents, these structures can be generated from semantic information embedded in the source. For instance, a table of contents can be built by reading the titles of chapters and sections. DITA provides elements to describe common document semantics.

Subject Matter Semantics

In some cases, the semantics of the content relate directly to the subject matter that the content describes. For instance, DITA supports tags that allow you to mark a piece of text as the name of a window in a software application (`wintitle`), or to mark a piece of text as applying only to a particular product.

Audience Semantics

In some cases, the semantics of the content relate to the audience that it is addressed to. For instance, a topic might be addressed to a particular role, or to a person with a particular level of experience. DITA provides an `audience` element to capture audience metadata.

Creating Topic Structures

Oxygen XML Editor provides a number of tools to help you create topic structures:

- The **Content Completion Assistant**, which shows you which elements can be created at the current position
- The **Model** view, which shows you the complete structure supported by the current element
- The **Outline** view, which shows you the current structure of your document
- The **DITA** toolbar, which helps you insert many common structures

Editing DITA Topics

Oxygen XML Editor provides a number of features to help you edit DITA topics.

Author Mode

DITA is an [XML format](#), although you do not have to write raw XML to create and edit DITA topics. Oxygen XML Editor provides a graphical view of your topics in [Author mode](#). Your topics will likely open in **Author** mode by default, so this is the first view you will see when you open or edit a DITA topic. If your topic does not open in **Author** mode, just click **Author** at the bottom left of the editor window to switch to this mode.

Author mode presents a graphical view of the document you are editing, similar to the view you would see in a word processor. However, there are some differences, including:

- **Author** mode is not a **WYSIWYG** view. It does not show you exactly what your content will look like when printed or displayed on-screen. The appearance of your output is determined by the DITA publishing process, and your organization may have modified that process to change how the output is displayed. Oxygen XML Editor has no way of determining what your final output will look like or where line breaks or page breaks will fall. Treat **Author** mode as a friendly visual editing environment, not a faithful preview of your output.
- Your document is still an XML document. **Author** mode creates a visual representation of your document by applying a CSS stylesheet to the XML. You can see the XML at any time by switching to **Text mode**. You, or someone in your organization, can change how the **Author** view looks by changing the CSS stylesheet or providing an alternate stylesheet.
- Your aim in editing a DITA document is not to make it look right, but to create a complete and correct DITA XML document. **Author** mode keeps you informed of the correctness of your content by highlighting XML errors in the text and showing you the current status in a box at the top right of the editor window. Green means that your document is valid, yellow means valid with warnings, and red means invalid. Warnings and errors are displayed when you place the caret on the error location.
- Your XML elements may have attributes set on them. Conventionally, attributes are used to contain metadata that is not displayed to the reader. By default, attributes are not displayed in the **Author** view (though there are some exceptions) and cannot be edited directly in the **Author** view (though in some cases the CSS that drives the display may use form controls to let you edit attributes directly). To edit the attributes of an element, place your cursor on the element and press **Alt+Enter** to bring up the attribute editor. Alternatively, you can use the **Attributes view** to edit attributes.

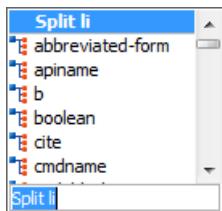


Tip: You can select **Hints** from the **Styles** drop-down list (available on the **Author Styles** toolbar) to display tooltips throughout the DITA document that offers additional information to help you with the DITA structure. For more information, see the [Selecting and Combining Multiple CSS Styles](#) section.

Content Completion Assistance

Because it is a structured format, DITA only allows certain elements in certain places. The set of elements allowed differ from one DITA topic type to another (this is what makes one topic type different from another). To help you figure out which elements you can add in any given place and help you understand what they mean, Oxygen XML Editor has a number of content completion assistance features.

- **The Enter key:** In **Author** mode, the Enter key does not create line breaks, it brings up the **Content Completion Assistant** to help you enter a new element. In XML, you do not use line breaks to separate paragraphs. You create paragraphs by creating paragraph elements (element `p` in DITA) and tools insert the line breaks in the output and on-screen.



The **Content Completion Assistant** not only suggests new elements you can add. If you hit **Enter** at the end of a block element (such as a paragraph) it suggests creating a new element of the same type. If you hit **Enter** in the middle of a block element, it suggests splitting that element into two elements.

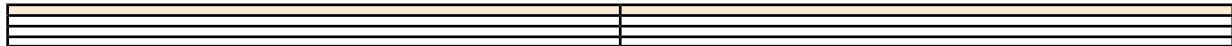
A useful consequence of this behavior is that you can create a new paragraph simply by hitting **Enter** twice (just as you might in a text editor).

As you highlight an element name, a basic description of the element is displayed. Select the desired element and hit **Enter** to create it.

To wrap an element around an existing element or piece of text, simply select it and hit **Enter** and use the **Content Completion Assistant** to choose the wrapper element.

- **The Model view:** You can see the entire model of the current element by opening the **Model** view (**Window > Show View > Model**, if the view is not already open). The **Model** view shows you what type of content the current element can contain, all the child elements it can contain, all its permitted

Table 2:



attributes, and their types.

- Tip:** You can also select **Inline actions** from the **Styles** drop-down list (available on the **Author Styles** toolbar) to display possible elements that are allowed to be inserted at various locations throughout the DITA document. For more information, see the [Selecting and Combining Multiple CSS Styles](#) section.

The DITA Toolbar

The **DITA** toolbar contains buttons for inserting a number of common DITA elements (elements that are found in most DITA topic types).



If the **DITA** toolbar is not displayed, right-click on the toolbar area and select it from the displayed list.

- Note:** The **DITA** toolbar contains a list of the most common elements and actions for DITA, such as inserting an image, creating a link, inserting a content reference, or creating a table. It does not contain a button for every possible DITA element. For a complete list of elements you can currently create, hit **Enter** to bring up the **Content Completion Assistant**.

The DITA Menu

Whenever the current document in the editor is a DITA document, the **DITA** menu is displayed in the menu bar. It contains a large number of commands for inserting elements, creating content references and keys, edit DITA documents, and controlling the display. These commands are specific to DITA and supplement the general editing commands available for all document types. As with the **DITA** Toolbar, the **DITA** menu does not list every possible DITA element.

Adding Images to a DITA Topic

There are several ways to add images to a DITA topic, depending on if you want to create a figure element (with a title and caption) or just insert an image inline, and if you want to use different versions of a graphic in various different situations. For instance, you might want to use a specific image for each different product version or output media.

Adding an Image Inline

Use the following procedure to add an image inline:

1. Place the caret in the position you want the graphic to be inserted.
2. Select the **Insert Image Reference** action. The **Insert Image** dialog box appears.

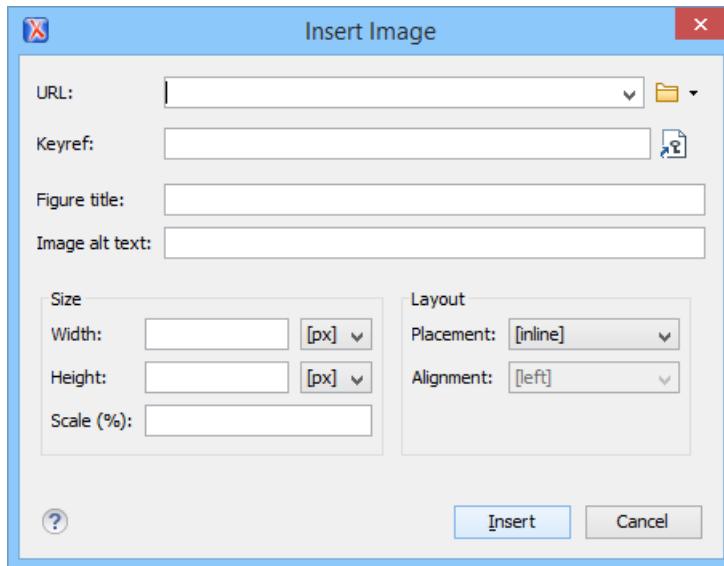


Figure 3: The Insert Image Dialog Box

- Configure the options in this dialog box and click **Insert**.

The **Insert Image** dialog box includes the following options for inserting images into a DITA document:

URL

Inserts an `image` element with an `href` attribute. You can type the URL of the image you want to insert or use the **Browse** drop-down list to select an image using one of the following options:

- Browse for local file** - Displays the **Open** dialog box to select a local file.
- Browse for remote file** - Displays the **Open URL** dialog box to select a remote file.
- Browse for archived file** - Opens the **Archive Browser** to select a file from an archive.
- Browse Data Source Explorer** - Opens the **Data Source Explorer** to select a file from a connected data source.
- Search for file** - Displays the **Find Resource** dialog box to search for a file.

Keyref

Opens the **Insert Key Reference** dialog box that presents the list of keys available in the current DITA Map. Use this dialog box to insert an `image` element with a `keyref` attribute. All keys that are presented in the dialog box are gathered from the root map of the current DITA map. Elements that have the `keyref` attribute set are displayed as links.



Note: If the DITA map is not opened in the **DITA Maps Manager** view, the **Insert Key Reference** dialog box does not display any keys.

Figure title

Use this text box to insert a `title` and `image` element inside a `fig` element.

Alternate text

Use this text box to insert an `alt` element inside the `image` element.

Size

Use this section to configure the **Width** and **Height** of the image, or **Scale** the image. Specifying a value in these options inserts a `width`, `height`, and `scale` attribute, respectively.

Layout

Use the options in this section to insert `placement` and `align` attributes into the `image` element.

Adding an Image in a Figure Element

To add an image in a figure:

1. Add a `fig` element to your document at the appropriate place.
2. Add a `title` and/or `desc` element to the `fig` element, according to your needs.
3. *Add an image element* to the `fig` element.

 **Note:** The `fig` element has a number of other child elements that may be appropriate to your content. See the [DITA documentation](#) for complete information on the `fig` element.

 **Note:** The order in which the `image`, `title`, and `desc` content are presented in output is determined by the output transformation. If you want to change how they are output, you may have to modify the output transformation, rather than your source content.

Adding an Image Using a Key Reference

If you want to use a different version of the image in various situations, such as screenshots for multiple platforms or different formats for various types of output media, you can reference the image using a key reference:

1. Create a DITA map to hold your image keys. You can create one map for each use or create a single map and profile the key definitions for multiple uses. For instance, you might create one map of images to be used in PDF and one for images to be used in Web output, or you might use the platform profiling attribute to manage different versions of a screenshot (one for Macintosh and another for the Windows version of your product).
2. For each image, create a `keydef` element with the following structure:

```
<keydef keys="image.test" href="img/test.png" format="png">
```

3. If you are using profiling, add the alternative `keydef` elements and the appropriate profiling attributes:

```
<keydef keys="image.test" href="img/win/test.png" platform="windows" format="png">
<keydef keys="image.test" href="img/mac/test.png" platform="mac" format="png">
```

4. If you are using separate maps, repeat in each map. For instance, if you are using a separate map for images in PDF output, add a `topic ref` to that map like this:

```
<keydef keys="image.test" href="img/test.eps" format="eps">
```

5. To insert an image by key, insert an `image` element and use a `keyref` attribute to point to the image:

```
<image keyref="image.test"/>
```

Oxygen XML Editor displays the image in [Author mode](#). Which image is displayed depends on the current profiling set that is applied and which [root map](#) is being used to resolve references.

6. Configure your build so that the appropriate image map is included for each output type and/or the appropriate profiling conditions are applied to each output.

 **Note:** You can also use the  [Insert Image Reference](#) action to insert a `keyref` attribute inside an `image` element.

Adding Tables to a DITA Topic

You can add a table to a DITA topic. By default, DITA supports two types of tables:

- DITA simple table model - This is the most commonly used model for basic tables.
- OASIS Exchange Table Model (a subset of the CALS table model) - This is used for more advanced functionality.

If you are using a specialized DITA vocabulary, it may contain specialized versions of these table models.

Because DITA is a structured format, you can only insert a table in places in the structure of a topic where tables are allowed. The Oxygen XML Editor DITA toolbar provides support for entering and editing tables. It also helps to indicate where you are allowed to insert a table or its components by graying out the appropriate buttons.

The **DITA** toolbar showing the insert table button available and the table editing buttons grayed out:



Figure 4: The DITA Toolbar

Inserting a DITA Simple Table

1. To insert a DITA simple table, click the table button on the **DITA** toolbar. The **Insert Table** dialog box appears.
2. Select the **Simple** model.
3. Select the appropriate options for rows and columns, creating a header, column widths, and framing of cells.
4. Click **Insert**. The table is inserted.

Inserting an OASIS Exchange Table Model (CALS) Table

To insert an OASIS Exchange Table, follow the same procedure as above, but choose the CALS model. Additional options for specifying row and column separators and alignment will be enabled.

When you insert a CALS table, you see a link for setting the `colspeccs` (column specifications) of your table. Click the link to open the `colspec` controls. Although they appear as part of the [Author view](#), the `colspec` link and the `colspec` controls will not appear in your output. They are just there to make it easier to adjust how the columns of your table are formatted.

▼ *A Sample CALS table*

▼ [colspecs...](#)

column

name	<input type="text" value="description"/>	number	<input type="text" value="1"/>	width	<input type="text" value="340pt"/>	align	<input type="button" value="center"/>	<input checked="" type="checkbox"/> colsep	<input checked="" type="checkbox"/> rowsep
-------------	--	---------------	--------------------------------	--------------	------------------------------------	--------------	---------------------------------------	--	--

column

name	<input type="text" value="price"/>	number	<input type="text" value="2"/>	width	<input type="text" value="69pt"/>	align	<input type="button" value="right"/>	<input checked="" type="checkbox"/> colsep	<input checked="" type="checkbox"/> rowsep
-------------	------------------------------------	---------------	--------------------------------	--------------	-----------------------------------	--------------	--------------------------------------	--	--

Description	Price

Editing an Existing Table

You can edit the structure of an existing table using the table buttons on the **DITA** toolbar to add or remove cells, rows, or columns, and to set basic table properties. Additional attributes can be used to fine-tune the formatting of your tables by using the [Attributes view](#) ([Window > Show View > Attributes](#)). See the [DITA documentation](#) for a full explanation of these attributes.

Also, remember that underneath the visual representation, both table models are really just XML, and, if necessary, you can edit the XML directly by switching to [Text mode](#).

Documents and Maps

In the DITA standard architecture you create documents by collecting topics into maps.

DITA Maps

A *DITA map* organizes a set of topics into a hierarchy. In most output formats, the structure of the map becomes the structure of the table of contents. Oxygen XML Editor provides support for creating and managing DITA maps through the **DITA Maps Manager**.

To watch a video on DITA editing and **DITA Maps Manager** view, go to http://oxygenxml.com/demo/DITA_Editing.html and http://oxygenxml.com/demo/DITA_Maps_Manager.html, respectively.

Book Maps

A *book map* is a specialized type of DITA map intended for creating the structure of a book.

Sub-Maps

You do not have to create an entire publication using a single map. It is generally good practice to break up a large publication into several smaller *sub-maps* that are easier to manage. You can include a sub-map in a main map using a `mapref` element. You can reuse sub-maps in a variety of different publications by including them in each of the different publication's main maps. The **DITA Maps Manager** provides support for creating and managing sub-maps.

Chunking

By default, many output types place a single topic on each output page. In some cases you may want to *output multiple topics as a single output page*. To support this, DITA provides the `chunk` attribute on the `topicref` element in a map.

Validating a Map

You should *validate your maps* to make sure that the individual topics are valid and that the relationships between them are working. Oxygen XML Editor provides a validation function for DITA maps that performs a comprehensive validation of a map and its topics.

Adding Topics to a DITA Map

When you are working in DITA, there are two approaches that you can use to create topics and maps. You can start by creating topics first, and then assemble your finished topics into one or more documents by creating one or more maps, or you can start by creating a map and then adding new topics to it as you work.

The topics-first approach is generally more appropriate if you are intending to do a lot of content reuse, as it encourages you to think of each topic as an independent unit that can be combined with other topics in various ways. The map-first approach will be more familiar to you if you are used to creating books or manuals as a whole. Oxygen XML Editor supports both approaches.

A DITA map organizes content hierarchically, so you can add a topic as a child of the map root or of any item already in the map. Therefore, the first step to adding a topic to a map is always to choose the place it will be inserted into the map.

Adding Existing Topics to a Map

At the XML-level, a topic is added to a map by adding a reference to the map that points to the topic. There are a number of different reference types that you can use. The default is the `topicref` element. See the *DITA documentation* for the full range of reference elements and their uses. Oxygen XML Editor provides several different tools for inserting reference elements into a map:

Using the Insert Reference Dialog Box

The [Insert Reference dialog box](#) allows you to create each of the different reference types and fill in the most commonly used attributes. You can open the **Insert Reference** dialog box in a number of ways:

- By right-clicking on an item in the current map where you want to add the reference, selecting **Append child** or **Insert After**, and selecting the type of reference to enter
- If the topic you want to add is currently open in the editor, you can use the same procedure, but select **Reference to the currently edited file...**
- By selecting an item in the map and clicking the  **Insert Reference** button.
- By selecting  **Insert Reference...** from the **DITA Maps** menu

Dragging and Dropping a File into the DITA Maps Manager

You can add a topic to a DITA map by dragging and dropping the file into the **DITA Maps Manager**. You can drag and drop files from any of the following:

- Your OS's file system explorer
- The **Project** view
- The **Open/Find Resource** view

Adding topics this way will not open the **Insert Reference** dialog box, but you can adjust all the same properties by opening the **Edit Reference** dialog box. Right-click on the reference you want to edit and choose **Edit Properties....**

Adding a New Topic to a Map

To add a new topic to a map:

1. Right-click on the place in the current map where you want to add the new topic.
2. Select either **Append child** or **Insert After** and then select **New topic....**

Creating DITA Sub-Maps

You can break up a large DITA map into more manageable pieces by creating sub-maps. A sub-map is simply a DITA map that is included by another DITA map. There is no separate markup for a sub-map.

For example, if you are creating a book, you might use one sub-map for each chapter of the book. If you are reusing a set of topics in multiple publications, you might collect them into a map and reuse the map as a sub-map in multiple other maps, rather than referencing the topics individually from the new maps.

You add a sub-map to a map the same way that you would [add a new topic or insert an existing topic into a map](#), except you choose a map rather than a topic to create or add. When adding a sub-map, to a map, make sure that you use a `mapref` element or a `topicref` element with the `format` attribute set to `ditamap`. In most cases, Oxygen XML Editor takes care of this for you.

You can move sub-maps around in a map [*just as you would move a topic*](#).

Chunking DITA Topics

By default, when a [DITA map](#) is published to an online format, each topic becomes a separate page in the output. In some cases, you may wish to combine multiple source topics into one output page. For instance, you may wish to combine several types of information into a single page, or you may have chosen to create many small DITA topics for reuse purposes but feel they are too small to be useful to a reader by themselves.

To chunk DITA topics, you set the chunking attribute on the `topicref` that contains the sub-topics in a DITA map. There are a number of different values you can set on the chunking attribute. See the [DITA documentation](#) for full details. To achieve the effects you want in your topics and table of contents, you may also need to set the `toc` and `collection-type` attributes on the sub-topics or container topic to suitable values. See the DITA documentation for details.

You can set the `collection-type` attribute on your topics using the **Edit Properties...** action in the **DITA Maps Manager**. To set the `toc` and `chunk` attributes, you must open the map file in the editor and add or edit the attributes directly (double-click on the map icon  in the **DITA Maps Manager** to open the map in the editor).

Manage a DITA Map

You may want to manage your DITA maps in a number of ways. For instance, you may want:

- to change the order and nesting of topics in a map
- to add or remove topics from the map
- to add keys to the topics in the map
- to add profiling (conditions) to the items in the map
- to change other properties of the items in the map

Changing the Order and Nesting of Topics in a Map

You can change the order and nesting of the topics in a map in several ways:

- By dragging and dropping topics within the **DITA Maps Manager**
- By highlighting a topic in the **DITA Maps Manager**, holding down the **Alt** key, and pressing the arrow keys
- By showing the extended **DITA Maps Manager** toolbar (press the settings icon  on the **DITA Maps Manager** toolbar and select the extended toolbar) and then using the arrow keys on the toolbar to move topics around on the map

Add and Remove Topics from a Map

You can [add](#) or remove topics from a map in a number of ways. Some ways to remove a topic from a map include:

- Highlight the topic and press **Delete**
- Highlight the topic and click the **Delete** button  on the **DITA Maps Manager** extended toolbar

Add Keys to Topic in a Map

You can [add keys to topics](#) in a map by right-clicking on an item in the map and choosing **Edit Properties....**

Profiling Sections of a Map

You can profile (make conditional) any section of a map by [adding one or more profiling attributes to the topics in a map](#). In the hierarchical structure of a map, any profiling applied to a topic that has children applies to those children as well. You cannot include a child topic if its parent topic is excluded.

Creating a Book in DITA

If you want to create a traditional book in DITA, you can use a book map to organize your topics into a book. A DITA book map is a specialized type of map, intended for creating output that is structured like a book. A book map allows you to add book-specific elements such as `frontmatter`, `part`, `chapter`, `appendix`, and `backmatter` to the map. How these book-specific elements are processed for publication is up to the processing script for each media. See the [DITA documentation](#) for details.

You can find additional support for creating books in DITA in the DITA for Publishers plugin, which is included with

.

To create a book in DITA using a book map:

1. Create a new book map. **File > New... > Framework templates > DITA Map > map > Bookmap**.
2. Create the structure of your book by adding the appropriate book sections and defining containers for chapters and any appendices. To add sections to a book map, or children to a section, right-click on the book map or section icon and choose **Append child**. The selections offered in the **Append child** and **Insert After** menus will adjust depending

on the element they are applied to. Consult the DITA documentation to fully understand the structure of a DITA book map and where to create each element.

3. Create special elements like an [index](#) and a [table of contents](#). The index and table of contents will be generated by the build process, based on the content of the map and the topics it points to.
4. [Add topics](#) to your chapters to add content to your book. You may find it easier to manage if you [use sub-maps](#) to create the content of your chapters. This keeps your book map from becoming long and difficult to manage.

Creating a Table of Contents in DITA

In DITA, the order and hierarchy of a document's table of contents is based directly on [the DITA map that defines the document](#). In many media, the creation of a table of contents, based on the map, is automatic. For example, you do not have to do anything special to create a table of contents in WebHelp output.

In other media, you need to tell DITA where the table of contents should occur. For example, in a book you need to tell DITA where to place the table of contents in the structure of the book, and whether to generate other common content lists, such as a list of figures or tables. You do this by [using a bookmap to define your book](#), and adding the appropriate elements to the `frontmatter`.

To configure a table of contents and other book lists:

1. Open your [bookmap](#) in the **DITA Maps Manager**.
2. Right-click on the book map icon and select **Append child > Frontmatter...**. The **Insert Reference** dialog box appears.
3. Click **Insert and Close** to insert the `frontmatter` element.
4. Right click on the `frontmatter` element and create a `booklists` element using **Append child > Book Lists....**
5. Use the same steps to create a `toc` element and to add any additional `booklists` elements you want, such as `tablelist`.

Creating an Index in DITA

In DITA, indexes are created from `indexterm` elements. You can insert index term elements:

- In the header of a topic. In paginated media, such as a printed book or a PDF, this results in an index entry that points to the page in which the topic starts, even if it is not the page in which the indexed term occurs.
- In the `topicref` element in a map that references the topic. This applies those index terms to that topic only when used in that map, allowing you to index topics differently in various publications. In paginated media, index entries point to the page in which the topic starts.
- In the body of a topic. In paginated media, this results in an index entry that points to the page in which the `indexterm` element occurs, even if that is not the page in which the topic starts.

To add index terms to the text of a topic of the topic header, [create the elements, as you normally would, in Oxygen XML Editor](#). To add index terms to a map, open the map in the editor and add the elements, as you normally would, in a topic.

In some media, indexes will be generated automatically when index entries are found in the source. For other media, such as books, you may need to tell DITA where to place the index. For instance, to add an index to a bookmap, you need to add an `indexlist` element to the `backmatter` of the book.

1. Open your [bookmap](#) in the **DITA Maps Manager**.
2. Right-click on the book map icon and select **Append child > Backmatter....**. The **Insert Reference** dialog box appears.
3. Click **Insert and Close** to insert the `backmatter` element.
4. Right-click on the `backmatter` element and create a `booklists` element using **Append child > Book Lists....**
5. Use the same steps to create an `indexlist` element.



Caution: Adding index entries and an `indexlist` to your project creates an instruction to the DITA publishing routines to create an index. There is no guarantee that all DITA output types or third-party

customizations obey that instruction or create the index the way you want it. Modifying the output may be necessary to get the result you want.

Validate a DITA Map

You should validate your maps regularly to make sure that your topics are valid, and all of the relationships between them are working. Changing one topic, image, or piece of metadata may create errors in references that rely on them. You may not discover these problems all at once. Validate your map to catch all of these kinds of problems. The longer you wait between validating your maps, the more difficult it may be to detect and correct any errors you find.

To validate a map:

1. In the **DITA Maps Manager**, make sure that the tab that holds your *root map* is selected and that the **Root map** selection is set either to the name of your *root map* or to <current map>.
2. Click the  **Validate and Check for Completeness** button on the **DITA Maps Manager** toolbar. The *Validate and Check for Completeness* dialog opens.
3. If you are using profiling, check the box **Use DITAVAL filters**, and select the appropriate options.
4. Select any other options you want to check.
5. Click **Check**.

The validator runs over the entire map. This process may take a while if the map is large. If there are any errors or warnings, they are displayed in separate pane. You can click on each error or warning to jump to the file where the problem was found.

Reuse

Reusing content is one of the key features of DITA. DITA provides a number of different methods for reusing content. Oxygen XML Editor provides support for each of these methods.

Reusing Topics

A DITA topic does not belong to any one publication. You add a DITA topic to a publication by referencing it in a map. You can reference the same topic in more than one map.

Reusing Content Elements

DITA allows you to reuse a content element by referencing it in another topic. DITA provides two mechanisms for including content by reference: `conref` and `conkeyref`. A `conref` creates a direct reference to a specific element of another topic. A `conkeyref` creates a reference to a key, which in turn points to a specific element of another topic. The advantage of using a `conkeyref` is that you can change the element that is included by changing the key reference. For example, since keys are defined in maps, a different key reference is used when you include your topic in a different map.

Oxygen XML Editor provides support for both `conref` and `conkeyref`.

While the `conref` and `conkeyref` mechanisms can be used to reference any content element, it is considered best practice to only `conref` or `conkeyref` content that is specifically set and managed as reusable content. This practice helps reduce expensive errors, such as an author accidentally deleting the source element that other topics are including by `conref`. Oxygen XML Editor can help you create a reusable component from your current content.

Reusing Content with Conditions

DITA allows you to conditionally profile parts of a topic so that certain parts of the topic are shown when certain profiling conditions are set. Profiling conditions can be set both within topics and in maps. When set in a topic, they allow you to suppress an element (such as paragraph), step in a procedure, item in a list, or even a phrase within a sentence. When set in a map, they allow you to suppress an entire topic or group of topics. You can then create a variety of different publications from a single map by applying profiling conditions to the build.

Reusing Content with Variables

DITA allows you to replace the content of certain elements with the value pointed to by a key. This mechanism effectively means that you can create variables in your content, which you can then output with various different values by changing the value the key points to. This is done by profiling the definition of the key value, or by substituting another map with various different key values that are defined.

Reusing DITA Topics

You can reuse a DITA topic simply by including it in more than one map:

1. Create a new map by clicking the **New...** document button and select **Framework templates > DITA Map** and then choose the appropriate type of map.
2. Add existing topics to the new map by dragging and dropping them from the **Project** view or the file system (or right-click on the map icon, or on a topic already in the map, and select **Append child...** or **Insert After...**).
3. If your topics use key references, set up the appropriate key definitions in your new map. Set the keys when you add the topics, or afterwards by right-clicking on the topic and select **Edit Properties....**
4. If you want to define relationships between topics, other than those defined in the topics themselves, *add a relationship table to your map* or to a separate map linked to your main map.
5. When you have finished adding topics, check that your map is complete and that all topic links and keys resolve correctly. To do this validation, click the  **Validate and Check for Completeness** action on the toolbar in the **DITA Maps Manager**.

Creating a DITA Content Reference

A DITA content reference, or `conref`, is a mechanism for inserting a piece of content from one topic into another topic. It is one of the main *content reuse features of DITA*.

In order for a `conref` to be created, the source content must have an `id` attribute that the `conref` can reference. Therefore, creating a `conref` may require both adding an `id` to the content to be reused and inserting a `conref` into the topic that reuses the referenced content.

To add an `id` to a DITA element in a topic, place the caret on the element and press **Alt+Enter** to bring up the in-place attribute editor. Enter `id` as the name of the attribute and a value of your choice in the value field. Note that the element may already have an `id` since in some cases Oxygen XML Editor generates an `id` value when the `id` attribute is created.

To create a content reference:

1. In **Author mode**, place the insertion point where you want the reused content to be inserted.
2. Click the  **Insert a DITA Content Reference** button on the DITA toolbar. The **Insert Content Reference** dialog box is displayed.
3. Select the **Element ID** of the element that you want to insert, and verify the content in the **Preview** pane.
4. Make any other selections you need in the **Insert Content Reference dialog box**.
5. Click **Insert and close**.

An alternate way to reuse content is to use the Oxygen XML Editor **Create Reusable Component...** and **Insert Reusable Component...** actions, which handle the details of creating an `id` and `conref` and creates reusable component files, which are separate from your normal content files. This can help you manage your reusable content more effectively.

You can also *insert reusable content using content key references*, which may make reusable content easier to manage.

Creating a DITA Content Key Reference

A DITA content key reference, or `conkeyref`, is a mechanism for inserting a piece of content from one topic into another. It is a version of the *DITA content reference mechanism* that uses `keys` to locate the content to reuse rather than direct references to topics that contain reused content.

As with a *conref*, a *conkeyref* requires that the element to be reused has an *id* attribute. It also requires the topic that contains the reusable content to be assigned a *key* in a map. As with all uses of keys, you can substitute multiple maps or *use profiling* to create more than one definition of keys in a single map.

This allows you to substitute different pieces of content for the same *conkeyref*. In other words, the same *conkeyref* can pull in content from various different sources, depending on how your build is configured. This can make it easier to create and manage sophisticated content reuse scenarios.

To add a content key reference to a topic:

1. In *Author mode*, place the caret where you want the reused content to be inserted.
2. Click the  **Insert a DITA Content Key Reference** button on the **DITA** toolbar. The **Insert Content Key Reference** dialog box is displayed.
3. Select the key for the topic that contains the content you want to reuse and press the **Subtopic** button.
4. Select the **Element ID** of the element that you want to insert, and verify the content in the **Preview** pane.
5. Make any other selections you need in the *Insert Content Reference dialog box*.
6. Click **Insert and close**.

Creating a Reusable Content Component

Almost any content element in DITA can be made reusable simply by adding an *id* attribute to it. The DITA content reference (*conref*) mechanism can reuse any element with an *id*. However, it is considered best practice not to reuse arbitrary pieces of text out of arbitrary topics due to the management overhead this creates, and the possibility of authors deleting or changing content that is reused in other topics without being aware that the content is reused. Instead, create and manage a separate set of topics that contain topics designed specifically for reuse.

Oxygen XML Editor makes it easy to create reusable content components from existing topic content.



Note:

To ensure that the topic file that contains the reusable component is a valid container for just the reusable content component, without having to include the other elements required by a standard topic type, Oxygen XML Editor creates a specialized topic type on the fly. This specialization is designed to make sure that the content is compatible with the topic type from which it is created.

1. Select the content you want to make into a reusable component.
2. Choose **DITA > Create Reusable Component...**.
The **Create Reusable Component** dialog box is displayed.
3. In the **Reuse Content** field, select the scope of the content to be made reusable. The choices presented are each of the container elements of the current caret position, allowing you to select how much of the current content you want to make reusable.
4. Add a description. This becomes the title of the topic that contains the reusable component, but is not part of the reusable content. It is just to help you identify the reusable content. It will not become part of your output.
5. Select whether you want the current content to be replaced by the reusable component. This is recommended, since the point of reuse is to maintain only one copy of the content.
6. Select a file name and location to save the topic containing the reusable component and click **Save**. It is considered best practice to save or store reusable components in an area set aside for that purpose.

If you selected to have your current content replaced by the reusable component, Oxygen XML Editor replaces the current content with a content reference. The content of the content reference will be shown in your current topic, but it will no longer be editable. To see the source of the reusable component, click on the  **Link** icon at the beginning of the inserted content.

You now have a reusable component that you can include in other topics using a *content reference* or a *content key reference*.

Insert a Reusable Content Component

You can insert a reusable content component that you [created using the Oxygen XML Editor Create Reusable Component ... action.](#)



Caution: This procedure is for inserting reusable components created using the Oxygen XML Editor **Create Reusable Component ...** action. It assumes certain things about the structure of the reusable content file that may not be true of reusable content created by other methods. It may not provide the expected results when used with files that do not have this structure.



Note:

The **Insert Reusable Content** action creates a DITA `conref` to insert the content, and creates a parent element for the `conref` attribute based on the type of the reusable element in the reusable component file. This action ensures that the correct element is used to create the `conref`. However, that element must still be inserted at a point in the current topic where that element type is permitted.

1. Place the insertion point where you want the reusable component inserted. Note that a reusable component must be inserted in a location that can accept the reusable content element.
2. Select **DITA > Insert Reusable Component....** The **Insert Reusable Component** dialog box is displayed.
3. Locate the reusable content file you want to use and click **Insert**.

Profiling (Conditional Content) in DITA

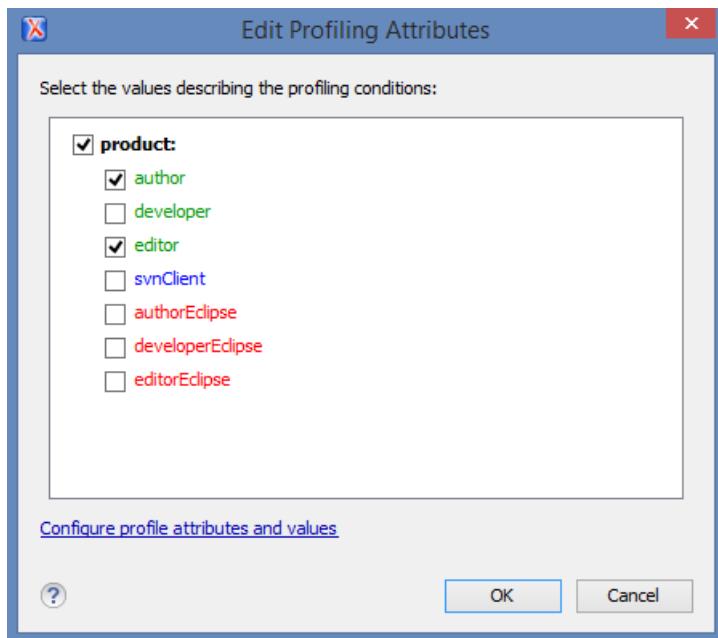
DITA provides a mechanism for profiling content (conditional). You can profile content elements or map elements by adding one or more of the default DITA profiling attributes (`product`, `platform`, `audience`, `rev`, `props`, and `otherprops`). You can also create your own [custom profiling attributes](#) and [custom profiling conditions sets](#). The profiling attributes may contain one or more tokens that represent conditions to be applied to the content when a publication is built.

For example, you could define a section of a topic that would only be included for a publication related to the Windows platform by adding the `platform` profiling attribute:

```
<section platform="windows">
```

Profiling Content

To apply a profiling attribute to an element in a topic or map, right-click on the element and select **Edit Profiling Attributes**. The **Edit Profiling Attributes** dialog box is displayed, allowing you to check each of the profiling tokens that apply for each attribute.



The profiling attributes, and their potential values, that appear in this dialog box depend on what has been configured in your copy of Oxygen XML Editor. The content of the dialog box is determined as follows:

- If your root DITA map references a DITA subject scheme map that defines values for the profiling attributes, those values are used. In the image above, which is taken from the Oxygen XML Editor documentation project, values are defined for seven different products, but none for other profiling attributes, which are therefore omitted from the dialog box.
- If your project defines *project level configuration values for the profiling attributes*, those values are used
- If your copy of Oxygen XML Editor defines global level *configuration values for the profiling attributes*, they are used
- Otherwise, a basic default set of profiling attributes and values are available

Visualizing Profiled Content

You can visualize the effect of profiling content by using the profiling tools in the Profiling/Conditional Text drop-down menu that is located on the **DITA Maps Manager** toolbar. You can select which profiles to show, or apply colors to text that is profiled in various ways, as shown in the following image:

Mowing equipment needs regular checks and maintenance. Monthly, you should:

- Refill the oil:
 - Remove the oil fill cap;
 - Pour new oil gradually. Regularly check the dipstick to see if the oil level reached the maximum mark;
- product [Gasoline]
- Sharpen the blades:
 - Clamp the blade to a vice or to the edge of a solid surface;
 - Using[»] an electric grinder or[«] audience [ExpertUser] a file, grind the length of the blade until it is sharp;
- Check the electric cable for any signs of wear. Replace it if worn; product [Electric]
- Clean the mower's underside for debris; product [Electric, Gasoline]
- Inspect the general state of the mower. Use a ratchet to tighten any loose bolts;
- Lubricate the gears of the manual lawn mower; product [Manual]

Creating Variable Text in DITA

You may often find that you want a certain piece of text in a topic to have a different value in various circumstances. For example, if you are reusing a topic about a feature that is shared between several products, you might want to make the name of the product variable so that the correct product name is used in the manual for each product.

For example, you might have a sentence like this:

```
The quick-heat feature allows [product-name] to come up to temperature quickly.
```

You need a way to substitute the correct product name for each product.

One way to do this would be to use conditional profiling, as in this figure:

```
<p>The quick-heat feature allows
<ph product="basic">Basic Widget</ph>
<ph product="pro">Pro Widget</ph>
<ph product="enterprise">Enterprise Widget</ph>
to come up to temperature quickly.</p>
```

Figure 5: Variable content using profiling

In DITA, you can create variable text using *keys*.

One way to do this would be to provide conditional values using the *product* profiling attribute.

However, this approach means that you are repeating the product names over and over again everywhere the product name is mentioned. This is time consuming for authors and will create a maintenance problem if the product names change.

The alternative is to use a key reference, as in the following example:

```
<p>The quick-heat feature allows <ph keyref="product"/> to come up to temperature quickly.</p>
```

Figure 6: Variable content using a key reference

The key reference stands in for the name of the product. When the content is published, the current value of the key *product* will be inserted.

Inserting a Key Reference

To insert a key reference into a document in Oxygen XML Editor **Author** mode:

1. Press **Enter** and select any DITA element that supports the `keyref` attribute.
2. Press **Alt+Enter** to bring up the attribute editor.
3. In the **Name** field, select `keyref`.
4. In the **Value** field, select or enter the name of the key.

Defining a Key

In DITA, keys are defined in maps, never in topics. A DITA map that defines various values of the `product` key would look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
    <!-- product name -->
    <keydef keys="product" product="basic">
        <topicmeta>
            <keywords>
                <keyword>Basic Widget</keyword>
            </keywords>
        </topicmeta>
    </keydef>
    <keydef keys="product" product="pro">
        <topicmeta>
            <keywords>
                <keyword>Professional Widget</keyword>
            </keywords>
        </topicmeta>
    </keydef>
    <keydef keys="product" product="enterprise">
        <topicmeta>
            <keywords>
                <keyword>Enterprise Widget</keyword>
            </keywords>
        </topicmeta>
    </keydef>
```



Note: The profiling of the names that was shown in the [Figure 5: Variable content using profiling](#) on page 78 figure is now contained in the map, where it only has to occur once.

Depending on which profiling conditions are active in Oxygen XML Editor, you will see different values of the key displayed in **Author** mode.

Linking

DITA provides support for various types of linking between topics, some of which is automated, while some is specified by the author. Oxygen XML Editor provides support for all forms of linking in DITA.

Linking Between Parent, Child, and Sibling Topics

A DITA map creates a hierarchical relationship between topics. That relationship map expresses a narrative flow from one topic to another, or it may be used as a classification system to help the reader find topics based on their classification, without creating a narrative flow. Because the relationship between topics in a hierarchy can be different in this way, you may want to create links between topics in various different ways. For instance, if your topics are supposed to be organized into a narrative flow, you may want to have links to the next and previous topics in that flow. If your topics are part of a hierarchical classification, you may want links from parent to child topics, and vice versa, but not next and previous links.

Parent, child, and sibling links are created automatically by the DITA output transformations (and may differ between various output formats). The kinds of links that are created are determined by the `DITA collection-type` attribute, which you can set in the **DITA Maps Manager** by selecting a topic and **Edit properties...** from the contextual menu.

Linking Between Related Topics

In addition to the relationships between topics that expressed by their place in the hierarchy of a map, a topic may be related to other topics in various ways. For instance, a task topic may be related to a concept topic that gives the background

of the task, or to a reference topic that provides data needed to complete the task. Task topics may also be related to other tasks in a related area, or concepts to related concepts.

Typically, they are grouped in a list at the end of the topic, though this depends on the behavior of the output transformation. DITA provides two mechanisms for expressing relationships between topics at the topic level: the `related-links` section of a topic, and relationship tables in maps.

Oxygen XML Editor provides tools for inserting `related-links` and relationship tables.

Linking in the Text of a Topic

DITA supports linking within the text of a topic using the `xref` element. The destination of the link can be expressed directly using the `href` attribute or indirectly using the `keyref` attribute. If you use the `keyref` attribute, you link to a key rather than directly to a topic. That key is then assigned to a topic in a map that includes that topic. This means that you can change the destination that a key points to either by profiling the key definition in the map or by substituting a different map in the build.

Oxygen XML Editor provides support for creating both direct links and key links, assigning keys to topics in a map, and profiling maps.

Managing Links

Links can break for a number of reasons. The topic that a link points to may be renamed or removed. A topic may be used in a map that does not include a topic it links to. A topic or a key may not exist in a map when a particular profile is applied. The **DITA Maps Manager** provides a way to validate all the links in the document, or documents described by a map. This can include checking with each of a set of profiling conditions applied.

Hierarchical Linking in DITA

To create hierarchical linking between the topics in a DITA map, you set the appropriate value of the `collection-type` attribute on the map. See the [DITA documentation](#) for the meaning of each of the values of the `collection-type` attribute.

 **Note:** The decision for when and how to create hierarchical links is made by the publishing scripts. The `collection-type` attribute does not force a particular style of linking. Rather, it declares what the nature of the relationship is between the topics. The publishing scripts use that information to decide how to link topics. Scripts for different types of media may make different decisions depending on what is appropriate for the media. You can provide additional instructions to the scripts using the `linking` attribute.

To add the `collection-type` to an item in a map:

1. Right-click on the topic and choose **Edit Properties...**. The **Edit Reference** dialog box is displayed.
2. Select the appropriate value from the **Collection type** drop-down list.

The **Edit Reference** dialog box does not include the `linking` attribute. To add a `linking` attribute:

1. Right-click on the map icon in the **DITA Maps Manager** and select **Open Map in Editor**. The map is opened in the editing area.
2. You can add the `linking` attribute in [**Author mode**](#) by placing the caret on the topic reference and pressing **Alt+Enter**, or you can edit it in [**Text mode**](#) by adding the attribute to the appropriate reference element (a `topic-ref` element in most cases).

Linking with Relationship Tables in DITA

A relationship table is used to express relationships between topics outside of the topics themselves. The DITA publishing scripts can then create links between related topics when the content is published.

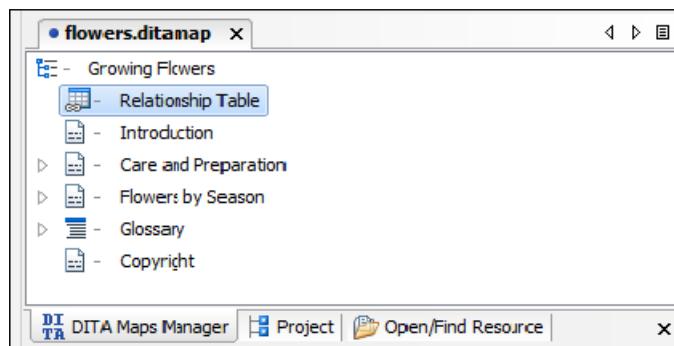
The reason for using a relationship table is to help make topics easier to reuse. If a topic links directly to another topic, this creates a dependency between the topics. If one topic is reused in a publication where the other is not used, the link is broken. By defining relationships between topics in a relationship table, you avoid creating this dependency.

To create an appropriate set of links between topics in multiple publications, you can create a separate relationship table for each publication. If you are creating a variety of different publications by applying profiling conditions to a single map, you can also profile your relationship table.

Follow these steps to create a relationship table:

1. If the map is currently open in the **DITA Maps Manager**, double-click the map icon  to open the map in **Author** mode. If it opens in **Text** mode, click **Author** at the bottom left to switch to **Author** mode.
2. Move the insertion point inside the *map* root element (usually *map*, but it might be *bookmap*, or another specialization of the *map* element). The easiest way to do this is to click below the title of the map in the editor and then press the up arrow once. Confirm that you are inside the *map* root element by checking the breadcrumbs at the top left of the editor window. You should only see the name of the *map* root element.
3. Click the  **Insert a DITA reltable** button on the **DITA Maps Manager** toolbar. The **Insert Relationship Table** dialog box is displayed.
4. Make the appropriate selections in the **Insert Relationship Table** dialog box. See the [DITA documentation](#) for a full explanation of the relationship table format and its options. Note that you can change all the selections that you make here later by using the buttons on the **DITA Maps Manager** toolbar or by editing the underlying XML.
5. To add topic references to your relationship table, drag and drop topics from the **DITA Maps Manager** or the **Project** view into the appropriate cell in the relationship table.

Relationship tables are also displayed in the **DITA Maps Manager** view, along with the other elements in its DITA map.



You can open the DITA map to edit the relationship table by doing one of the following:

- Double-click the appropriate relationship table in the **DITA Maps Manager**.
- Select the relationship table in the **DITA Maps Manager** and press **Enter**.
- Select **Open** from the contextual menu of the relationship table in the **DITA Maps Manager**.

Linking with a related-links Element in DITA

To create a list of related links within the topic you are writing (as opposed to [specifying relationships externally with a relationship table](#)):

1. Open the topic in **Author** mode in which you want the link to occur.
2. Select one of the *related link* actions from the  **Link** drop-down list that is available on the **DITA** toolbar. You can choose between the following types of related links:
 - **Related Link to Topic...** - Opens an **Insert Reference** dialog box that allows you to select a reference.
 - **Related Link to File...** - Opens an **Input URL** dialog box that allows you to select a file.
 - **Related Link to Web Page...** - Opens an **Input URL** dialog box that allows you to select a web link.

This creates a related links section at the bottom of your topic. DITA provides the *linklist* and *linkpool* elements to help you refine how your links are created and processed. See the [DITA documentation](#) for details.

Inline Linking in DITA

You can create inline links in the content of a DITA topic using the `xref` element. To insert a link in Oxygen XML Editor *Author mode*, use the actions available in the  - **Link** drop-down list from the **DITA Toolbar**.

Linking Using Keys

To make links easier to manage, you may choose to use *keys* for linking, rather than creating direct links to topics. By using keys, you avoid creating a direct dependency between topics, which can make it easier to reuse topics in various publications. It can also be helpful in verifying the completeness of a publication, by ensuring that a publication map provides a key definition for every key reference used in the content.

Links based on keys require two pieces. The first piece is a key definition, which assigns a key to a topic so that other topics can link to it. The second piece is a key reference, which is created in an `xref` element and specifies the key to link to. Thus, the key reference points to a key definition, and the key definition points to a topic.

Key definitions are created in maps, as an element on the `topicref` element that points to a topic. Thus, a key reference points to a key definition on a `topicref` that points to a topic.

This allows you to assign a particular key to one topic in one map and to another topic in another map. When a topic that links to that key is used in each of these maps, the links work correctly in both maps.



Note: You can define more than one key on the same topic reference. Topics can link to any of the keys. This allows you to have topics that link to various keys that are linked to the same topic in one map and to other topics in another map.

Adding a Key to a Map

To add a key to a map:

1. In the **DITA Maps Manager**, right-click on the topic reference to which you want to add a key and choose **Edit Properties**. The **Edit Reference** dialog box is displayed.
2. Enter the key value in the **Keys** field.
3. Make sure to save the changes to the DITA map before attempting to insert the key reference in your topic.

Creating a Link Using a Key

To create a link using a key:

1. In *Author mode*, highlight the text you want to be a link.
2. Select **Key Reference...** from the  - **Link** drop-down list that is available in the **DITA Toolbar**. The **Insert Key Reference** dialog box is displayed.
3. Use the filter field to find the key you want to link to, then select it and click **Insert and close**. The link is created in your topic.

Output

In DITA, you create output by running a transformation on a DITA map. Transformations for various types of output are provided by the DITA Open Toolkit. Oxygen XML Editor provides support for configuring and running transformation using transformation scenarios.



Note: Oxygen XML Editor does not create any output formats itself. Oxygen XML Editor runs externally defined transformations that produce output, and displays the result in the appropriate application, but the output itself is produced by the external transformation, not by Oxygen XML Editor.

Customizing Outputs

You can customize the appearance of any of the output types by customizing the output transformations. There are two ways to do this:

1. Most transformations are configurable by passing parameters to the transformation script. Oxygen XML Editor allows you to set parameters on a transformation scenario and save the parameters for future use. It can also allow you to prompt for a parameter whenever a transformation scenario is run. You can set up more than one transformation scenario for a given output type, which allows you to maintain several customized transformation scenarios for different types of output configurations.
2. If you want to customize an output in a way not supported by the customization options, you can create a modified version of the transformation code. Some transformation scripts export specific forms of extension or customization. You should consult the SPFE Open Toolkit for the transformation type that you are interested in to see what customization options it supports. Oxygen XML Editor provides full editing and debugging support from XSLT and CSS stylesheets, which you can use to modify transformation code.

You can also write your own output transformation scripts to produce a type of output not supported by the DITA Open Toolkit. Oxygen XML Editor provides a full development environment for developing such transformations. You can create Oxygen XML Editor transformation scenarios to run these scripts once they are complete.

Generating Output from DITA Content

As a structured writing format, DITA produces structured content (content that is annotated with specific structural and semantic information rather than with formatting information). To create a publication, your DITA map and its associated topics must be processed by a transformation script. That script is responsible for how the structural and semantic information in the DITA files is converted into formatting information for display.

This means that you can display the same DITA content in various different ways, media, or publications. It also means that you cannot control every aspect of the presentation of your content in your DITA files. The only way to change the formatting is to change the transformation routines that create it.

Therefore, to create output from your DITA content you have to run a transformation on your content. Oxygen XML Editor provides a mechanism called transformation scenarios to help you configure and run transformations.

To select and run a transformation scenario on your map:

1. Click the  **Configure Transformation Scenario(s)** button. The **Configure Transformation Scenario(s)** dialog box appears. This dialog box lists all the transformation scenarios that have been configured in your project. Oxygen XML Editor provides a default set of transformation scenarios, but the people in charge of your DITA system may have provided others that are specifically configured for your needs.
2. Select the transformation scenario you want to run and click **Apply Associated**. The transformation scenario runs in the background. You can continue to work in Oxygen XML Editor while the transformation is running. If there are errors or warnings, Oxygen XML Editor displays them when the transformation is complete. If the transformation is successful, Oxygen XML Editor opens the output in the appropriate application.
3. To rerun the same scenario again, click the  **Apply Transformation Scenario(s)** button.

Metadata

Metadata is a broad concept that describes data that explains or identifies other data. Metadata can be used for many purposes, from driving automation of document builds to enabling authors and readers to find content more easily. DITA provides a number of different types of metadata, each of which has different uses and is created in different places and ways. Some of the most important forms of metadata in DITA are topic and taxonomy.

Topic Metadata

Topic metadata describes the topic and what it is about. Topic metadata can be inserted in the `prolog` element of a topic or inside the `topicref` element that points to a topic from a map. In other words, metadata about the topic can be asserted by the topic itself, or can be assigned to it by the map that includes it in the build. This allows various different

maps to assign metadata to the same topic. This may be appropriate where you want to describe a topic differently in various documents.

Taxonomy and Subject Scheme

A taxonomy is a controlled vocabulary. It can be used to standardize how many things in your content and metadata are named. This consistency in naming can help ensure that automated processes work correctly, and that consistent terminology is used in content, and in metadata. In DITA, taxonomies are created using subject scheme maps. When you are authoring, many of the values you choose from have been defined in subject scheme maps.

Background: Keys

DITA uses keys to insert content that may have different values in various circumstances. Keys provide a way to reference something indirectly. This can make it easier to manage and to reuse content in a number of ways.

You can think of keys like renting a post office box. Instead of the mail going directly from the sender to your house, it now goes to the post office box. You then go to the post office box and bring the mail back to your house. If you move to a new house, your mail still gets to you because it comes to the same post office box. You do not have to send change of address cards to all the people who send you mail. Your mailbox address is the key that makes sure your mail always reaches you, even if you move.

Similarly, if you use keys in your content to reference other content, you do not have to update the source content in order to change the value of the key or what it points to. You just change the definition of the key.

Keys are thus a very general mechanism. DITA uses keys for referencing different kinds of content for various purposes.

Keys for Values

You can use keys to represent *values that may vary in different outputs*. For instance, you may have several products that share a common feature. When you want to describe that feature, you need a way to insert the name of the product, even though that name is different depending on which product the feature description is being used for.

Keys for Topics

You can assign a key to a topic and use that key to reference that topic for various purposes, such as reuse or linking. As always, keys are defined in maps, so the key definition is done using the keys attribute of the `topicref` element:

```
<topicref href="quick-heat.dita" keys="feature.quick-heat"/>
```

Once a key is assigned to a topic, you can use it to reference that topic for various purposes:

- You can *create a link* to it using `<xref keyref="feature.quick-heat" />`. This allows you to change the target of the link by changing the topic that is pointed to by the key (for example, by profiling).
- You can use it *in a map to create a reference to a topic* by key: `<topicref keyref="feature.quick-heat" />`. This allows you to change which topic is inserted in the map by the build, by changing the topic that is pointed to by the key.
- You can use it to *insert a content reference*. In this case, the content reference uses the key to locate the topic to pull content from. It uses a `conkeyref` attribute: `<procedure conkeyref="feature.quick-heat/preheat-procedure" />`. In this example, `feature.quick-heat` is the key, and `preheat-procedure` is the id of a procedure within the topic for that key. Using this mechanism, you could have multiple versions of the preheat procedure in various topics and control which one is inserted by changing the topic that is pointed to by the key.

Keys for Graphics

You can assign a key to an image (using a map to point to the image file) and *insert the image using the key*.

Chapter

5

Perspectives

Topics:

- *Perspectives*
- *Dockable Views and Editors*
- *Help Menu*

This chapter describes the editing perspectives of Oxygen XML Editor.

Perspectives

The Oxygen XML Editor interface uses standard interface conventions and components to provide a familiar and intuitive editing environment across all operating systems.

With Oxygen XML Editor, you can edit documents in one of the following perspectives:

Editor perspective

Documents editing is supported by specialized and synchronized editors and views.

XSLT Debugger perspective

XSLT stylesheets can be debugged by tracing their execution step by step.

XQuery Debugger perspective

XQuery transforms can be debugged by tracing their execution step by step.

Database perspective

Multiple connections to relational databases, native XML databases, WebDAV sources and FTP sources can be managed at the same time in this perspective: database browsing, SQL execution, XQuery execution and data export to XML.

Editor Perspective

To edit the content of your documents, use the **Editor** perspective (**Window > Open perspective > Editor**).

When two or more views are displayed, the application provides divider bars. Divider bars can be dragged to a new position increasing the space occupied by one panel while decreasing it for the other.

As the majority of the work process centers around the Editor area, other views can be hidden using the controls located on the views headers.

This perspective organizes the workspace in the following sections:

- **Main menu** - Provides menu driven access to all the features and functions available in Oxygen XML Editor.
- **Main toolbar** - Provides easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function.
- **Editor area** - The place where you spend most of your time, reading, editing, applying markup and validating your documents.
- **Outline view** - It provides an XML document overview and offers functions such as modifications follow-up, document structure change, document tag selection, elements filtering.
- **Model view** - Presents the current edited element structure model and additional documentation as defined in the schema.
- **Results view** - Displays result messages obtained by performing user operations like search (the results of the **Find All** action applied to the current file or the results of a find action applied on a set of files), *validation*, *transformation* and *spell check*. The following actions are available in a vertical toolbar on the right side of the view:

Grouping options

Allows you to choose grouping criteria, which reflects into a flat or tree-like presentation of the results. To display the results in a flat layout, you can either uncheck all selected criteria or press the  **Ungroup all** button. All these actions are also available in the table header contextual menu.

Highlight all results in editor

Displays or hides all the highlights.

Remove selected

Removes the currently selected messages from the list.

✿ Remove all

Clears the message list.

A contextual menu available on the table grid allows you to:

- Navigate to previous and next message. As alternative, you can use the default shortcut keys: **Ctrl Shift]** (**Command Shift]** on OS X) for navigating to the next and **Ctrl Shift [** (**Command Shift [** on OS X) for navigating to the previous message.
- Print the results or save them either in text or XML format.
- Expand or collapse all displayed items (with the shortcuts **Alt Shift PageUp** for the **Expand All** action and **Alt Shift PageDown** for the **Collapse All** action).
- Restore default grouping criteria and table column widths.
- **Project view** - Enables the definition of projects and logical management of the documents they contain.
- **Attributes view** - Presents all possible attributes of the current element
- **Elements view** - Presents a list of all defined elements that you can insert at the current caret position according to the document's schema.
- **Entities view** - Displays a list with all entities declared in the current document as well as built-in ones.
- **Transformation Scenarios view** - Displays a list with all currently configured transformation scenarios.

The Results View

The **Results View** displays the messages generated as a result of user actions like validations, transformations, search operations and others. Each message is a link to the location related to the event that triggered the message. Double clicking a message opens the file containing the location and positions the cursor at the location offset. The actions that can generate result messages are:

- *Validate action*
- *Transform action*
- *Check Spelling in Files action*
- *Find All* action from *the Find/Replace dialog*
- *Find/Replace in Files dialog*
- *Search References action*
- *XPath expression results*
- *SQL results*

Description - 12 items	Resource	Location
▲ D:\Projects\UserGuide\DTA\topics\batch-transformation.dita (2 items)		
href="results-view.dita#results-view" format="dita">>Results View</xref>. All entries in the Results View point to the location of the code that triggered them. </p>	batch-transformation.dita	29:61
	batch-transformation.dita	30:7
▲ D:\Projects\UserGuide\DTA\topics\editor-perspective.dita (1 item)		
<uicontrol>Results view</uicontrol> - Displays result messages obtained by performing user	editor-perspective.dita	52:22
▲ D:\Projects\UserGuide\DTA\topics\xpath-view.dita (1 item)		
<title>The XPath Results View</title>	xpath-view.dita	12:30
Find in Files - Result... <input type="button" value="X"/>		

Figure 7: Results View

The view provides a toolbar with the following actions:

✿ Group by actions

A set of actions that group the messages according to a selected criteria so that they can be presented in a hierarchical layout. The criteria used for grouping can be the severity of the errors (error, warning, info message and so on), the resource name, the description of the message and so on. The **Ungroup all** action removes the grouping rule so that the messages are presented as a continuous list.



Highlight all results in editor

Oxygen XML Editor highlights all matches obtained after executing an XPath expression, or performing one of the following operations: **Find All**, **Find in Files**, **Search References**, and **Search Declarations**. Click **Highlight all results in editor** again to turn off highlighting.



Note: To customize highlighting behavior, [open the Preferences dialog box](#) and go to **Editor > Highlights category**. You can do the following customizations:

- set a specific color of the highlights depending on the type of action you make.
- set a maximum number of highlights that the application displays at any given time.

Remove actions

The **X Remove selected** and *** Remove all** reduce the number of messages from the view by removing them.

The actions available on the contextual menu are:

Show message

Displays a dialog box with the full error message, which is useful for a long message that does not have enough room to be displayed completely in the view.

↑ Previous message

Moves the selection in the view to the message above the current one.

↓ Next message

Moves the selection in the view to the message below the current one.

✖ Remove selected

Removes selected messages from the view.

* Remove all

Removes all messages from the view.



Copy

Copies the information associated with the selected messages:

- the file path of the document that triggered the output message,
- the path of the main file (in case of *validation scenario* it is the path of the file from which the validation starts and which can be different than the validated file),
- error severity (error, warning, info message and so on.),
- name of validating processor,
- name of *validation scenario*,
- the line and column in the file that triggered the message.

Select All

Extends the selection to all the messages from the view.

Print Results ...

Sends the complete list of messages to a printer. For each message the included details are the same as the ones for [the Copy action](#).

Save Results ...

Saves the complete list of messages in a file in text format. For each message the included details are the same as the ones for [the Copy action](#).

Save Results as XML

Saves the complete list of messages in a file in XML format. For each message the included details are the same as the ones for [the Copy action](#).

Group by

A set of actions that group the messages according to a selected criteria so that they can be presented in a hierarchical layout. The criteria used for grouping can be the severity of the errors (error, warning, info message and so on), the resource name, the description of the message and so on. The **Ungroup all** action removes the grouping rule so that the messages are presented as a continuous list.

Ungroup all

Removes the grouping rule set by a **Group by** action so that the errors are presented as a continuous list.

Show Group Columns

Displays/hides columns used as grouping criteria.

Restore Defaults

Restores the column size for each column and the grouping rule that were saved in the user preferences the last time when this view was used (possible in the previous Oxygen XML Editor session). If it is the first time when this view is used, the action sets an initial default column size for each column and a grouping rule which is appropriate for the type of messages, for example:

- group the messages by the path of the validated file in case of error messages from a validation action or spelling errors reported by the action **Check Spelling in Files**,
- no grouping rule for the results of *applying an XPath expression*.

Expand All

Expands all the nodes of the tree, which is useful when the messages are presented in a hierarchical mode.

Collapse All

Collapses all the nodes of the tree, which is useful when the messages are presented in a hierarchical mode.

XSLT Debugger Perspective

The XSLT Debugger perspective (**Window > Open perspective > XSLT Debugger**) allows you to detect problems in an XSLT transformation by executing the process step by step. The workspace is organized as an editing area assisted by special helper views. The editing area contains editor panels, *allowing you to split it horizontally or vertically* in a stack of XML editor panels and a stack of XSLT editor panels. The XML files and XSL files can be edited in *Text mode* only.

The layout of the XSLT Debugger perspective is composed of the following components:

- *Control toolbar* - Contains all actions needed in order to configure and control the debug process.
- *Source document view* - Displays and allows editing of data or document oriented XML files (documents).
- *Stylesheet document view* - Displays and allows editing of XSL files (stylesheets).
- *Output document view* - Displays the transformed output that results from the input of a selected document (XML) and selected stylesheet (XSL) to the transformer. The result of transformation is dynamically written as the transformation is processed. There are three types of views for the output: a text view (with XML syntax highlight), an XHTML view, and one text view for each `xsl:result-document` element used in the stylesheet (if it is a XSLT 2.0 / 3.0 stylesheet).
- *Information views* - Distributed in two panes, they are displaying various types of information that can be used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This allows you to obtain a clear view of the transformation progress.

You are able to add XPath expression automatically in the **XWatch** view using the **Watch expression** action from the contextual menu. In case you select an expression or a fragment of it and then click **Watch expression** in the contextual menu, the entire selection is presented in the **XWatch** view. Using **Watch expression** without selecting an expression displays the value of the attribute from the caret position in the **XWatch** view. Variables detected at the caret position are also displayed.



Note: Expressions displayed in the **XWatch** view are *normalized* (unnecessary white spaces are removed from the expression).

XQuery Debugger Perspective

The XQuery Debugger perspective (**Window > Open perspective > XQuery Debugger**) resembles [the XSLT Debugger perspective](#). You can use it to detect problems in an XQuery transformation process by executing the process step by step in a controlled environment and inspecting the information provided in the special views.

The workspace is organized as follows:

- Source document view - allows editing of data or document-oriented XML files (documents).
- XQuery document view - allows editing of XQuery files.
- Output document view - displays the transformed output that results from the input of a selected document (XML) and selected XQuery document to the XQuery transformer. The result of transformation is dynamically written as the transformation is processed. There are two types of views for the output: a text view (with XML syntax highlight) and an XHTML view.
- Control toolbar - contains all actions you need for configuring and controlling the debug process.
- Information views - distributed in two panes they are displaying various types of information that can be used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views.

You are able to add XPath expression automatically in the **XWatch** view using the **Watch expression** action from the contextual menu. In case you select an expression or a fragment of it and then click **Watch expression** in the contextual menu, the entire selection is presented in the **XWatch** view.

 **Note:** Expressions displayed in the **XWatch** view are normalized (unnecessary white spaces are removed from the expression).

To watch our video demonstration about the XQuery debugging capabilities in Oxygen XML Editor, go to http://www.oxygenxml.com/demo/XQuery_Debugger.html.

Database Perspective

The Database perspective (**Window > Open perspective > Database**) allows you to manage a database, offering support for browsing multiple connections at the same time, relational and native XML databases, SQL execution, XQuery execution and data export to XML.

This perspective offers database specific support for:

- Oracle Berkeley DB XML Database
- eXist XML Database
- IBM DB2 (Enterprise edition only)
- JDBC-ODBC Bridge
- MarkLogic (Enterprise edition only)
- Microsoft SQL Server 2005 and Microsoft SQL Server 2008 (Enterprise edition only)
- MySQL
- Oracle 11g (Enterprise edition only)
- PostgreSQL 8.3 (Enterprise edition only)
- Documentum xDb (X-Hive/DB) 10 XML Database (Enterprise edition only)
- Documentum (CMS) 6.5 (Enterprise edition only)

The XML capabilities of the databases marked in this list with "Enterprise edition only" are available only in the Enterprise edition of Oxygen XML Editor. The non-XML capabilities of any database listed here are available also in the Academic and Professional editions of Oxygen XML Editor by registering the database driver as a generic JDBC driver (the *Generic JDBC* type in the list of driver types) when [defining the data source](#) for accessing the database in Oxygen XML Editor.

The non-XML capabilities are:

- browsing the structure of the database instance
- opening a database table in the **Table Explorer** view
- handling the values from **XML Type** columns as String values

The XML capabilities are:

- displaying an XML Schema node in the tree of the database structure (for databases with such an XML specific structure) with actions for opening/editing/validating the schemas in an Oxygen XML Editor editor panel
- handling the values from columns of type XML Type as XML instance documents that can be opened and edited in an Oxygen XML Editor editor panel
- validating an XML instance document added to an XML Type column of a table, etc.

For a detailed feature matrix that compares the Academic, Professional and Enterprise editions of Oxygen XML Editor please [go to the Oxygen XML Editor website](#).

 **Note:** Only connections configured on relational data sources can be used to import data to XML or to generate XML schemas.

The perspective provides the following functional areas:

- Main menu - provides access to all the features and functions available within Oxygen XML Editor.
- Main toolbar - provides easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function.
- Editor area - the place where you spend most of your time, reading, editing, applying markup and validating your documents.
- Data Source Explorer - provides browsing support for the configured connections.
- Table explorer - provides table content editing support for inserting new rows, deleting table rows, cell value editing, export to XML file.

Dockable Views and Editors

All the Oxygen XML Editor views available in the [Editor Perspective](#), [XSLT Debugger Perspective](#), and [XQuery Debugger Perspective](#) are dockable.

You can drag any view to any margin of another view or editor inside the Oxygen XML Editor window. Once you create a layout that suites your needs, you are able to save it from **Window > Export Layout...**. Oxygen XML Editor creates a layout file containing the preferences of the saved layout. To load a layout, go to **Window > Load Layout**. To reset it, select **Window > Reset Layout**.

 **Note:** The **Load Layout** menu lets you select between the default layout, a predefined layout, or a custom layout. The changes you make using the **Load Layout** menu are also reflected in the **Application Layout** preferences page.

The changes you make to any layout are preserved between working sessions. Also, changing to a different layout and returning to the previous one does not alter the changes you made to the first layout. The predefined layout files are saved in the [preferences directory](#) of Oxygen XML Editor.

To gain more editing space in the Oxygen XML Editor window, click  **Toggle auto-hide** in any view. This button sets the view in the *auto-hide* state, making it visible only as a vertical tab, at the margins of the Oxygen XML Editor window. To display a view in the *auto-hide* state, hover its side-tab with your cursor, or click it to keep the view visible until you click elsewhere. A view can also be set to a floating state, making it independent from the rest of the Oxygen XML Editor window.

You can drag the editors and arrange them in any order you like, both horizontally and vertically.

The next figure presents two editors arranged as horizontal tiles. To arrange them vertically, drag one of them on top of the other. In the example below, the `personal.xml` file was dragged over the `personal-schema.xml` file. When doing this, a dark grey rectangle marks the rearranged layout.

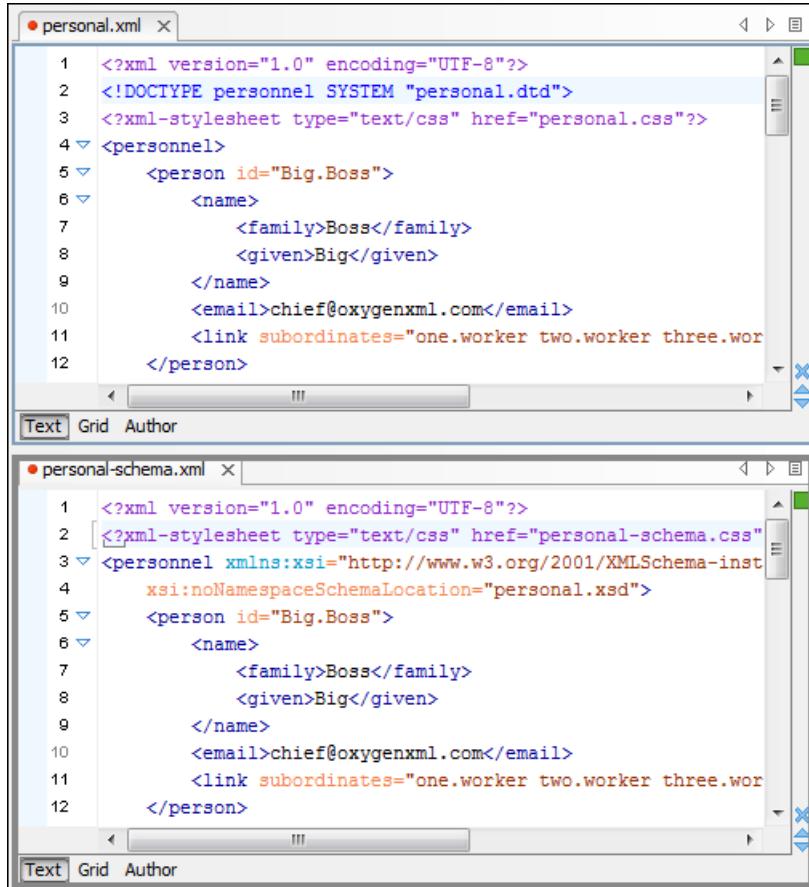


Figure 8: Drag and Drop Editors

You can also tile or stack all open editors, both in the *Editor Perspective* or in the *Database Perspective*, using the following actions from the **Editor** toolbar or **Window** menu:

Tile Editors Horizontally

Splits the editing area into horizontal tiles, one for each open file.

Tile Editors Vertically

Splits the editing area into vertical tiles, one for each open file.

Stack Editors

The reverse of the **Tile Editors Horizontally/Vertically** actions. Stacks all open editors.



Note: When tiled, you can still drag and drop the editors, but note that they are docked in the same way as a window/view (instead of just tabs). You are actually rearranging the editor windows, so drag the editor tab and drop it to one of the sides of an editor (left/right/top/bottom). While dragging, you will see the grey dark rectangle aligned to one of the sides of the editor, or around the entire editor window. If you drop it to one of the sides it will dock to that side of the editor. If you drop it when the rectangle is around the entire window of the editor it will get stacked on top of that editor. You can also grab one of the stacked editors and tile it to one of the sides.

To scroll through the tiled editors in the same time, enable the **Synchronous scrolling** action.

You can divide the editing area vertically and horizontally using the following actions available in the **Editor** toolbar and **Window** menu:

- **Split horizontally**
- **Split vertically**

-  **Unsplit**

To maximize or restore the editors, go to **Window > Maximize/Restore Editor Area**.

When the opened documents titles do not fit in the tab strip, the scroll wheel can be used to scroll the editor title tabs to the left or right the same way the two arrows on the right are acting. The following shortcuts can be used to switch between edited files: **Ctrl F6 (Command F6 on OS X)** and **Ctrl Shift F6 (Command Shift F6 on OS X)**. These shortcuts display a small popup window that cycles through all opened files.

The default layout of any of the *Editor Perspectives*, *XSLT Debugger Perspective* and *XQuery Debugger Perspective* can be restored at any time with the **Reset Layout** action found in the **Window** menu.

Any Oxygen XML Editor view or toolbar can be opened at any time from the **Window > Show View** and **Window > Show Toolbar** menus. The current (focused) dockable view is made invisible (switched to hidden state) using the **Ctrl Shift F4 (Command Shift F4 on OS X)** shortcut. The users who prefer to use the keyboard instead of the mouse may find this shortcut to be a faster way of closing a view than clicking the **Close** button from the title bar of the view. The complementary action (opening a view with a shortcut) requires setting a custom shortcut for each view in *the Menu Shortcut Keys preferences*.

To watch our video demonstration about dockable and floating views and editors in Oxygen XML Editor, go to http://oxygenvml.com/demo/Dockable_VIEWS.html.

Help Menu

The **Help** menu contains the following actions:

Welcome

Displays *the Welcome dialog box*. By default, this dialog box is also presented each time you start Oxygen XML Editor. If you have editors opened it is presented as an individual dialog box. Otherwise, it is integrated in the empty editing area.

Help (F1)

Opens the **Help** dialog box.

Use online help (Enabled by default)

If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Oxygen XML Editor attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

Show Dynamic Help view

Opens the **Dynamic Help** view and automatically loads the relevant help section for the focused editor, view, or dialog box.

Install new add-ons...

Allows you to select add-ons to install.

Check for add-ons updates...

Select this option to check for available add-on updates. Oxygen XML Editor lets you know if updates are available. If updates are available, select **Review updates** to open the **Manage add-ons** view.

Manage add-ons...

Select this option to open the **Manage add-ons** view.

Check for a New Version

Checks the availability of a new application version.

Browse oXygen Website

Opens the Oxygen XML Editor website in your default web browser.

Register...

Opens the registration dialog box.

Report problem...

Opens a dialog box that allows the user to write the description of a problem that was encountered while using the application. You are able to change the URL where the reported problem is sent by using the `com.oxygenxml.report.problems.url` system property. The report is sent in XML format through the `report` parameter of the POST HTTP method.

Support Center

Opens the Oxygen XML Editor Support Center web page in a browser.

Tip of the Day

Opens a dialog box that displays tips for using Oxygen XML Editor.

About

Opens the [About dialog box](#).

The About Dialog

The **About** dialog contains the following tabs:

- **Copyright** - this tab contains general information about the product and the version of the product you are using, along with contact details and the SGN number. Details regarding the memory usage are also presented at the bottom of the dialog
- **Libraries** - this tab presents the list of third party libraries that Oxygen XML Editor uses. To view the End User Licence Agreement of each library, double click it
- **Frameworks** - this tab contains a list with the frameworks that are bundled with Oxygen XML Editor
- **System Properties** - this tab contains a list with system properties and their values. The contextual menu allows you to select and copy the properties

The Welcome Dialog

The **Welcome** dialog is an informative and functional panel displayed when you start Oxygen XML Editor. It presents upcoming events, useful video demonstrations, the tip of the day and also gives you fast access to recently used files and projects and the possibility to create new ones.

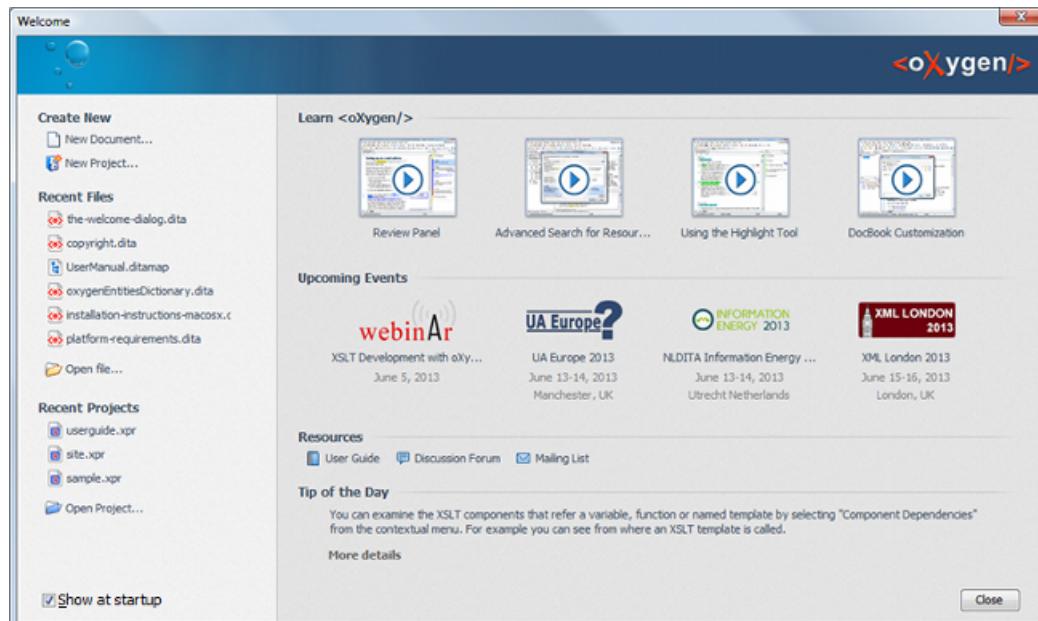


Figure 9: The Welcome Dialog

This dialog is displayed cut out from the main layout of Oxygen XML Editor if documents are already opened in the editing area at startup. If not, the **Welcome** dialog is integrated in the empty editing area. To display it any time you want, go to **Help > Welcome**.

Chapter

6

Editing Modes

Topics:

- [*Text Editing Mode*](#)
- [*Grid Editing Mode*](#)
- [*Author Editing Mode*](#)

To better suit the type of editing that you want to perform, Oxygen XML Editor offers the following modes:

- ***Text*** - this mode presents the source of an XML document.
- ***Grid*** - this mode displays an XML document as a structured grid of nested tables.
- ***Author*** - this mode enables you to edit in a WYSIWYG like editor.
- ***Design*** - this mode is found in the schema editor and represents the schema as a diagram.

Text Editing Mode

The **Text** mode of Oxygen XML Editor provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, etc. These actions are executed from the menu bar or toolbar and also by invoking their usual keyboard shortcuts.

The Undo/Redo Actions

The typical undo and redo actions are available in Oxygen XML Editor:

 **Undo** Ctrl Z (Command Z on OS X) - menu **Edit** >  **Undo**

Reverses a maximum of 200 editing actions (configurable in the [Editor preferences page](#)) to return to the preceding state.

 **Note:** Complex operations like **Replace All** or **Indent selection** count as single undo events.

 **Redo** Ctrl Y (Command Z on OS X) - menu **Edit** >  **Redo**

Recreates a maximum of 100 editing actions that were undone by the **Undo** function.

Copying and Pasting Text

The typical copying and pasting actions are available:

Edit >  **Cut** Ctrl C (Command C on OS X)

Removes the current selected node from the document and places it in the clipboard as RTF. All text attributes such as color, font, or syntax highlight are preserved when pasting into another application.

Edit >  **Copy** Ctrl C (Command C on OS X)

Places a copy of the current selection in the clipboard as RTF. All text attributes such as color, font, or syntax highlight are preserved when pasting into another application.

Edit >  **Paste** Ctrl V (Command V on OS X)

Places the current clipboard content into the document at the cursor position.

Edit > **Select All** Ctrl A (Command A on OS X)

Selects the entire body of the current document, including whitespace preceding the first and following the last character.

Finding and Replacing Text in the Current File

This section walks you through the find and replace features available in Oxygen XML Editor.

You can use a number of advanced views depending on what you need to find in the document you are editing or in your entire project. The [Find/Replace dialog box](#) allows you to search through the current project or selected resources and offers a set of options to improve your search. The [Find All Elements/Attributes dialog box](#) allows you to search through the structure of the current document for elements and attributes.

As an alternative to the dedicated search operations, you can also use the [Quick Find toolbar](#).

The Find/Replace Dialog Box

To open the **Find/Replace** dialog box, use the  **Find/Replace...** action that is available in the **Find** menu, on the toolbar, or by pressing Ctrl F (Command F on OS X).

You can use the **Find/Replace** dialog box to perform the following operations:

- Replace occurrences of target defined in the **Find** area with a new fragment of text defined in **Replace with** area.
- Find all the occurrences of a word or string of characters (that can span over multiple lines) in the document you are editing. This operation also takes into account all the white spaces contained in the fragment you are searching for.



Note: The **Find/Replace** dialog box counts the number of occurrences of the text you are searching for and displays it at the bottom of the dialog box, above the **Close** button. This number is also displayed in [the Results view](#).

The *find* operation works on multiple lines, meaning that a find match can cover characters on more than one line of text. To input multiple-line text in the **Find** and **Replace with** areas, do one of the following:

- Press **Ctrl Enter (Command Enter on OS X)** on your keyboard.
- Use the **Insert newline** contextual menu action.

You can use [Perl-like regular expressions syntax](#) to define patterns. A content completion assistant window is available in the **Find** and **Replace with** areas to help you edit regular expressions. It is activated every time you type \ (backslash key) or on-demand if you press **Ctrl Space (Command Space on OS X)** on your keyboard.

The *replace* operation can bind regular expression capturing groups (\$1, \$2, etc.) from the find pattern.

To replace the tag-name start tag and its attributes with the new-tag-name tag use as **Find** the expression <tag-name(\s+)(.*)> and as **Replace with** the expression <new-tag-name\$1\$2>.

The dialog box contains the following options:

- **Find** - The target character string to search for. You can search for Unicode characters specified in the \uNNNN format. Also, hexadecimal notation (\xNNNN) and octal notation (\0NNNN) can be used. In this case you have to select the **Regular expression** option. For example, to search for a space character you can use the \u0020 code.
- **Replace with** - The character string with which to replace the target. The string for replace can be on a line or on multiple lines. It can contain Perl notation capturing groups, only if the search expression is a regular expression and the **Regular expression** option is selected.



Note: Some regular expressions can block indefinitely the application. If the execution of the regular expression does not end in about 5 seconds, the application displays a dialog box that allows you to interrupt the operation.



Note: Special characters like *newline* and *tab* can be inserted in the **Find** and **Replace with** text boxes using dedicated actions in the contextual menu (**Insert newline** and **Insert tab**).

Unicode characters in the \uNNNN format can also be used in the **Replace with** area.

- The **History** button - Contain lists of the last find and replace expressions. Use the **Clear history** action from the bottom of the lists to remove these expressions.
- **XPath** - The XPath 2.0 expression you input in this combo is used for restricting the search scope.



Note: The **Content Completion Assistant** helps you input XPath expressions, valid in the current context.

- **Direction** - Specifies if the search direction is from current position to end of file (**Forward**) or to start of file (**Backward**).
- **Scope** - Specifies whether the **Find/Replace** operation is executed over the entire content of the edited document (**All** option), or over the selected lines of text (**Only selected lines** option). In case the selection spans across multiple lines, when you open the **Find/Replace** dialog box, the scope is set to **Only selected lines**.
- **Case sensitive** - When checked, the search operation follows the exact letter case of the text entered in the **Find** field.
- **Whole words only** - Only entire occurrences of a word are included in the search operation.
- **Incremental** - The search operation is started every time you type or delete a letter in the **Find** text box.
- **Regular expression** - When this option is enabled, you can use regular expressions in [Perl-like regular expressions syntax](#) to look for specific pieces of text.
 - **Dot matches all** - A dot used in a regular expression also matches end of line characters.
 - **Canonical equivalence** - If enabled, two characters will be considered a match if, and only if, their full *canonical* decompositions match. For example, the á symbol can be inserted as a single character or as two characters (the a character, followed by the tilde character). This option is disabled by default.

- **Wrap around** - When the end of the document is reached, the search operation is continued from the start of the document, until its entire content is covered.
- **Enable XML search options** - This option is only available when editing in **Text** mode. It provides access to a set of options that allow you to search specific XML component types:
 - **Element names** - Only the element names are included in the search operation which ignores XML-tag notations ('<', '/', '>'), attributes or white-spaces.
 - **Element contents** - Search in the text content of XML elements.
 - **Attribute names** - Only the attribute names are included in the search operation, without the leading or trailing white-spaces.
 - **Attribute values** - Only the attribute values are included in the search operation, without single quotes(') or double quotes(").
 - **Comments** - Only the content of comments is included in the search operation, excluding the XML comment delimiters ('<!--', '-->').
 - **PIs (Processing Instructions)** - Only the content are searched, skipping '<?', '?>'. e. g.: <?processing instruction?>
 - **CDATA** - Searches inside content of CDATA sections.
 - **DOCTYPE** - Searches inside content of DOCTYPE sections.
 - **Entities** - Only the entity names are searched.

The two buttons **Select All** and **Deselect All** allow a simple activation and deactivation of all types of XML components.



Note: Even if you enable all options of the **Enable XML search options** section, the search is still XML-aware. If you want to perform the search over the entire file content, disable **Enable XML search options**.

- **Find All Elements...** - Available when editing in **Author** mode, you can use this link to extend the search scope to XML-specific markup (names and values of both attributes and elements).
- **Find** - Executes a find operation for the next occurrence of the target. It stops after highlighting the find match in the editor panel.
- **Replace** - Executes a replace operation for the target followed by a find operation for the next occurrence.
- **Find All** - Executes a find operation and displays all results in the **Results view**. The results are *displayed in the Results view*.
- **Replace All** - Executes a replace operation in the entire scope of the document.
- **Replace to End** - Executes a replace operation starting from current target until the end of the document, in the direction specified by the current selection of the **Direction** switch (**Forward** or **Backward**).

The Find All Elements Dialog Box

To open the **Find All Elements** dialog box, go to **Find > Find All Elements... (Ctrl Shift E (Command Shift E on OS X))** or from the shortcut **Find All Elements** that is available in *the Find / Replace dialog box*. It assists you in defining XML element / attribute search operations in the current document.

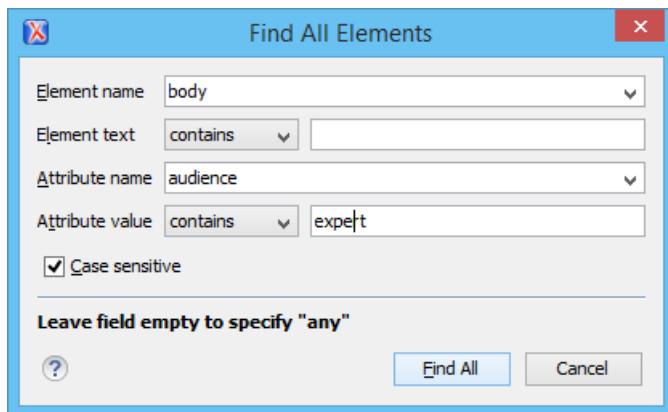


Figure 10: Find All Elements Dialog Box

The dialog box can perform the following actions:

- Find all the elements with a specified name
- Find all the elements that contain, or does not contain, a specified string in their text content
- Find all the elements that have a specified attribute
- Find all the elements that have an attribute with, or without, a specified value

You can combine all of these search criteria to filter your results.

The following fields are available in the dialog box:

- **Element name** - the qualified name of the target element to search for. You can use the drop-down list to find an element or enter it manually. The drop-down list is populated with valid element names collected from the associated schema. To specify *any* element name, leave the field empty.

 **Note:** Use the qualified name of the element (<namespace prefix>:<element name>) when the document uses this element notation.
- **Element text** - the target element text to search for. The drop-down list beside this field allows you to specify that you are looking for an exact or partial match of the element text. For *any* element text, select **equals** in the drop-down list and leave the field empty. If you leave the field empty but select **equals** in the drop-down list, only elements with no text will be found. Select **not equals** to find all elements that do not include the specified text.

 **Note:** Use the qualified name of the attribute (<namespace prefix>:<attribute name>) when the document uses this attribute notation.
- **Attribute name** - the name of the attribute that must be present in the element. You can use the drop-down list to select an attribute or enter it manually. The drop-down list is populated with valid attribute names collected from the associated schema. For *any* or no attribute name, leave the field empty.
- **Attribute value** - the drop-down list beside this field allows you to specify that you are looking for an exact or partial match of the attribute value. For *any* or no attribute value, select **equals** in the drop-down list and leave the field empty. If you leave the field empty but select **equals** in the drop-down list, only elements that have at least an attribute with an empty value will be found. Select **not equals** to find all elements that have attributes without a specified value.

 **Note:** Use the qualified name of the attribute (<namespace prefix>:<attribute name>) when the document uses this attribute notation.
- **Case sensitive** - when this option is checked, operations are case-sensitive

When you press **Find All**, Oxygen XML Editor tries to find the items that match all the search parameters. The results of the operation are presented as a list in the message panel.

The Quick Find Toolbar

A reduced version of *the Find / Replace dialog* is available as *a dockable toolbar*. To display it press the **Alt Shift F** key combination or invoke the **File >  Quick Find...** action. By default, the toolbar is displayed at the bottom of the

Oxygen XML Editor window, above the status bar, but can be changed at any time by dragging (and docking) it to a different location. To hide the toolbar, use the  Close button or **ESC** key.

All matches are highlighted in the current editor.

The toolbar offers the following controls:

- A search input box where you insert the text you want to search for. The input box keeps a history of the last used search text. The background color of the input box turns red when no match is found.
- **Next** and **Previous** buttons. They allow you to advance to the next or previous match.
- **All** button. Highlights in the current document all matches of the search string.
- **Incremental** check box. The search operation is started every time you type or delete a character in the search input box.
- **Case sensitive** check box. When selected, the search operation follows the exact letter case of the search text.
- Shortcuts to the  [Find/Replace](#) and  [Find/Replace in Files](#) dialog boxes.

Keyboard Shortcuts for Finding the Next and Previous Match

Navigating from one match to the next or previous one is very easy to perform using the **F3** and **Shift F3** keyboard shortcuts. They are useful to quickly repeat the last find action performed in [the Find / Replace dialog](#), taking into account the same find options.

Regular Expressions Syntax

Oxygen XML Editor uses the Java regular expression syntax. It is **similar** to that used in Perl 5, with several exceptions. Thus, Oxygen XML Editor does not support the following constructs:

- The conditional constructs `(?{X})` and `(?(condition)X|Y)`.
- The embedded code constructs `(?{code})` and `(??{code})`.
- The embedded comment syntax `(?#comment)`.
- The preprocessing operations `\1`, `\u`, `\L`, and `\U`.

Other notable difference:

- In Perl, `\1` through `\9` are always interpreted as back references; a backslash-escaped number greater than 9 is treated as a back reference if at least that many sub-expressions exist, otherwise it is interpreted, if possible, as an octal escape. In this class octal escapes must always begin with a zero. In Java, `\1` through `\9` are always interpreted as back references, and a larger number is accepted as a back reference if at least that many sub-expressions exist at that point in the regular expression, otherwise the parser will drop digits until the number is smaller or equal to the existing number of groups or it is one digit.
- Perl uses the `g` flag to request a match that resumes where the last match left off.
- In Perl, embedded flags at the top level of an expression affect the whole expression. In Java, embedded flags always take effect at the point at which they appear, whether they are at the top level or within a group; in the latter case, flags are restored at the end of the group just as in Perl.
- Perl is forgiving about malformed matching constructs, as in the expression `*a`, as well as dangling brackets, as in the expression `abc]`, and treats them as literals. This class also accepts dangling brackets but is strict about dangling meta-characters like `+`, `?` and `*`.

Finding and Replacing Text in Multiple Files

To open the **Find/Replace in Files** dialog box, use the  [Find/Replace in Files](#) action that is available in the following locations::

- The **Find** menu.
- The main toolbar.
- The contextual menu of the **DITA Maps Manager** view.
- The contextual menu of the **Project** view.

- The contextual menu of the **Data Source Explorer** view for Documentum xDb (X-Hive/DB), eXist and WebDAV connections. This action is available for Documentum (CMS), but lacks the *replace* feature.

The operation works on both local and remote files from an (S)FTP, WebDAV or [CMS](#) server.

It enables you to define *Search for* or *Search for and Replace* operations across a number of files. You can use [Perl-like regular expression syntax](#) to match patterns in text content. The *replace* operation can bind regular expression capturing groups (\$1, \$2, etc.) from the find pattern.

To replace the tag-name start tag and its attributes with the new-tag-name tag use as **Text to find** the expression <tag-name(\s+)(.*> and as **Replace with** the expression <new-tag-name\$1\$2>.

The encoding used to read and write the files is detected from the XML header or from the BOM. If a file does not have an XML header or BOM Oxygen XML Editor uses by default the UTF-8 encoding for files of type XML, that is for files with one of the extensions: .xml, .xsl, .fo, .xsd, .rng, .nvd1, .sch, .wsdl or [an extension associated with the XML editor type](#). For the other files it uses [the encoding configured for non-XML files](#).

You can cancel a long operation at any time by pressing the **Cancel** button of the progress dialog box displayed when the operation is executed.

Because the content of read-only files cannot be modified, the **Replace** operation is not processing those files. For every such file, a warning message is displayed in the message panel.

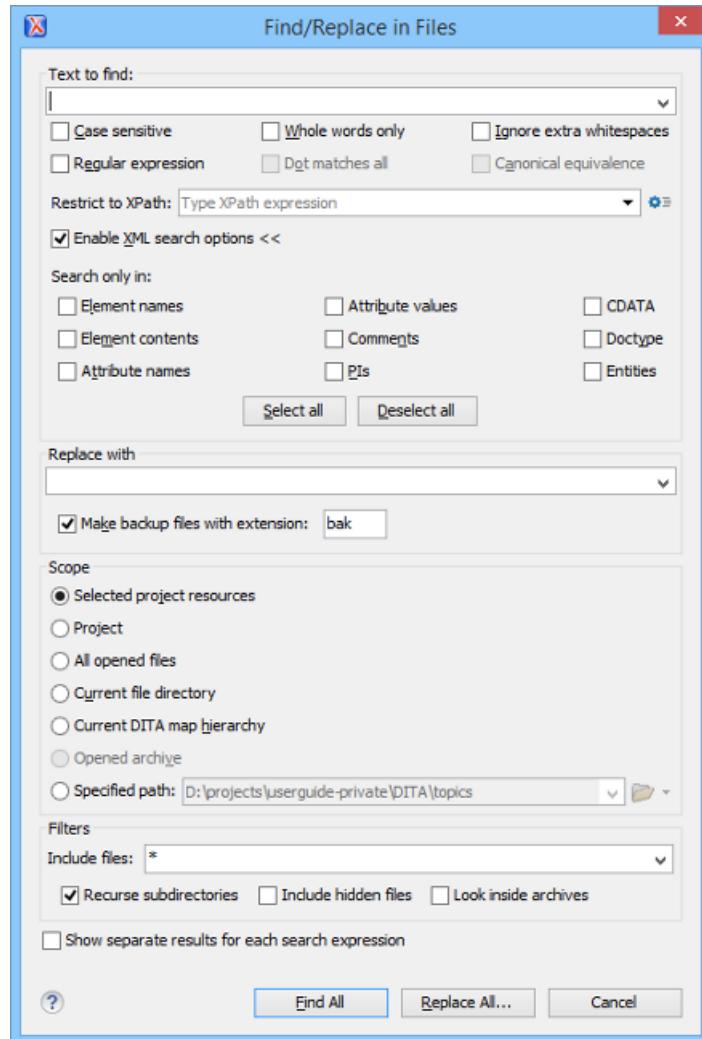


Figure 11: Find / Replace in Files Dialog Box

The dialog box contains the following options:

- **Text to find** - The target character string to search for. You can search for Unicode characters specified in the \uNNNN format. Also, hexadecimal notation (\xNNNN) and octal notation (\0NNNN) can be used. In this case you have to select the **Regular expression** option. For example, to search for a space character you can use the \u0020 code.
 - **Case sensitive** - When checked, the search operation follows the exact letter case of the value entered in the **Text to find** field.
 - **Whole words only** - Only entire occurrences of a word are included in the search operation.
 - **Ignore extra whitespaces** - If enabled, the application normalizes the content (collapses any sequence of whitespace characters into a single space) and trims its leading and trailing whitespaces when performing the search operation.
 - **Regular expression** - When this option is enabled, you can use regular expressions in *Perl-like regular expressions syntax* to look for specific pieces of text.
 - **Dot matches all** - A dot used in a regular expression also matches end of line characters.
 - **Canonical equivalence** - If enabled, two characters will be considered a match if, and only if, their full *canonical decompositions* match. For example, the **á** symbol can be inserted as a single character or as two characters (the **a** character, followed by the tilde character). This option is disabled by default.
 - **XPath** - The XPath 2.0 expression you input in this combo is used for restricting the search scope.
-  **Note:** The **Content Completion Assistant** helps you input XPath expressions, valid in the current context.
- **Enable XML search options** - This option is only available when editing in **Text** mode. It provides access to a set of options that allow you to search specific XML component types:
 - **Element names** - Only the element names are included in the search operation which ignores XML-tag notations ('<', '/', '>'), attributes or white-spaces.
 - **Element contents** - Search in the text content of XML elements.
 - **Attribute names** - Only the attribute names are included in the search operation, without the leading or trailing white-spaces.
 - **Attribute values** - Only the attribute values are included in the search operation, without single quotes(') or double quotes(").
 - **Comments** - Only the content of comments is included in the search operation, excluding the XML comment delimiters ('<!--', '-->').
 - **PIs (Processing Instructions)** - Only the content are searched, skipping '<?', '?>'. e. g.: <?processing instruction?>
 - **CDATA** - Searches inside content of CDATA sections.
 - **DOCTYPE** - Searches inside content of DOCTYPE sections.
 - **Entities** - Only the entity names are searched.

The two buttons **Select All** and **Deselect All** allow a simple activation and deactivation of all types of XML components.



Note: Even if you enable all options of the **Enable XML search options** section, the search is still XML-aware. If you want to perform the search over the entire file content, disable **Enable XML search options**.

- **Replace with** - The character string with which to replace the target. It may contain regular expression group markers if the search expression is a regular expression and the **Regular expression** checkbox is checked.
- **Make backup files with extension** - In the replace process Oxygen XML Editor makes backup files of the modified files. The default extension is .bak, but you can change the extension as you prefer.
- **Selected project resources** - Searches only in the selected files of the currently opened project. This option is not displayed when this dialog box is opened from the contextual menu of the **DITA Maps Manager view** and **Archive Browser** view.
- **Project files** - Searches in all files from the current project. This option is not displayed when this dialog box is opened from the contextual menu of the **DITA Maps Manager view** and **Archive Browser** view.

- **All opened files** - Searches in all files opened in Oxygen XML Editor (regular files or DITA Maps). You are prompted to save all modified files before any operation is performed. This option is not displayed when this dialog box is started from the contextual menu of the [DITA Maps Manager view](#) and [Archive Browser](#) view.
- **Current file directory** - The search is done in the directory of the file opened in the current editor panel. If there is no opened file, this option is disabled. This option is not displayed when this dialog box is opened from the contextual menu of the [DITA Maps Manager view](#) and [Archive Browser](#) view.
- **Current DITA Map hierarchy** - The search is done in all maps and topics referenced by the currently selected DITA map, opened in the [DITA Maps Manager](#) view.
- **Opened archive** - The search is done in an archive opened in the [Archive Browser](#) view. Displayed only when this dialog box is opened from the [Archive Browser](#) view.
- **Specified path** - Chooses the search path.
- **Include files** - Narrows the scope of the operation only to the files that match the given filters.
- **Recurse subdirectories** - When enabled, the *pretty print* is performed recursively for the specified scope. The one exception is that this option is ignored if the scope is set to **All opened files**.
- **Include hidden files** - When enabled, the search is also performed in the hidden files.
- **Include archives** - When enabled, the search is also done in all individual file entries from all supported ZIP-type archives.
- **Show separate results for each search expression** - When enabled, the application opens a new tab to display the result of each new search expression. When the option is unchecked, the search results are displayed in the *Find in Files* tab, replacing any previous search results.
- **Find All** - Executes a find operation and returns the result list to the message panel. The results are [displayed in a view](#) that allows grouping the results as a tree with two levels.
- **Replace All** - Replaces all occurrences of the target contained in the specified files.

When you replace a fragment of text, Oxygen XML Editor provides a preview of the changes you make. The **Preview** dialog box is divided in two sections. The first section presents a list of all the documents containing the fragment of text you want to modify. The second section offers a view of the original file and a view of the final result. It also allows you to highlight all changes using the vertical bar from the right side of the view. The **Next change** and **Previous change** buttons allow you to navigate through the changes displayed in the **Preview** dialog box.



Caution: Use this option with caution. Global search and replace across all project files does not open the files containing the targets, nor does it prompt on a per occurrence basis, to confirm that a replace operation must be performed. As the operation simply matches the string defined in the find field, this may result in replacement of matching strings that were not originally intended to be replaced.

Changing the Font Size

The font size of the editor panel can be changed with the following actions:

- **Document > Font size > Increase editor font ([Ctrl NumPad+ \(Command NumPad+ on OS X\)](#))** - increases the font size with one point for each execution of the action.
- **Document > Font size > Decrease editor font ([Ctrl NumPad- \(Command NumPad- on OS X\)](#))** - decreases the font size with one point for each execution of the action.
- **Document > Font size > Normal editor font ([Ctrl 0 \(Command 0 on OS X\)](#))** - resets the font size to *the value of the editor font set in Preferences*.

Word/Line Editor Actions

The **Text** editor implements the following actions:

[**Ctrl Delete \(Command Delete on OS X\)**](#)

Deletes the next word.

[**Ctrl Backspace \(Command Backspace on OS X\)**](#)

Deletes the previous word.

Ctrl W (Command W on OS X)

Cuts the previous word.

Ctrl K (Command K on OS X)

Cuts to end of line.

Dragging and Dropping the Selected Text

To move a whole region of text to other location in the same edited document just select the text, drag the selection by holding down the left mouse button and drop it to the target location.

You can also copy content from other applications and paste it into the document. For more information on this feature, see [the Smart Paste Support section](#).

Inserting a File at Caret Position

The action from menu **Document > File > Insert file...** inserts the content of the file with the specified file path in the current document at the current position of the caret.

Opening the File at Caret in System Application

The **Document > File > Open File at Caret in System Application** action opens the file (identified by its link) or web page (identified by a web link) found at caret position. The target is open in the default system application associated with that file type.

Opening the File at Caret Position

The action from menu **Document > File > Open File at Caret** opens in a new panel the file with the file path at the caret position. If the path represents a directory path, it will be opened in system file browser. If the file does not exist at the specified location, the error dialog box that is displayed contains a **Create new file** button that starts the **New document** wizard. This allows you to choose the type or the template for the file. If the action succeeds, the file is created with the referenced location and name and is opened in a new editor panel. This is useful when you decide first on the file name and after that you want to create it in the exact location specified at the current caret position.

Printing a File

Printing is supported in **Text**, **Author**, and **Grid** modes of the XML editor panel. The action from menu **File > Print > Ctrl P (Command P on OS X)** displays the **Page Setup** dialog used for defining the page size and orientation properties for printing.

A **Print Preview** action is available in the **File** menu. It allows you to manage the format of the printed document.

 **Note:** If you are printing in the Author visual editing mode to change the styling of the document in the printed output you can use the [CSS print media type](#).

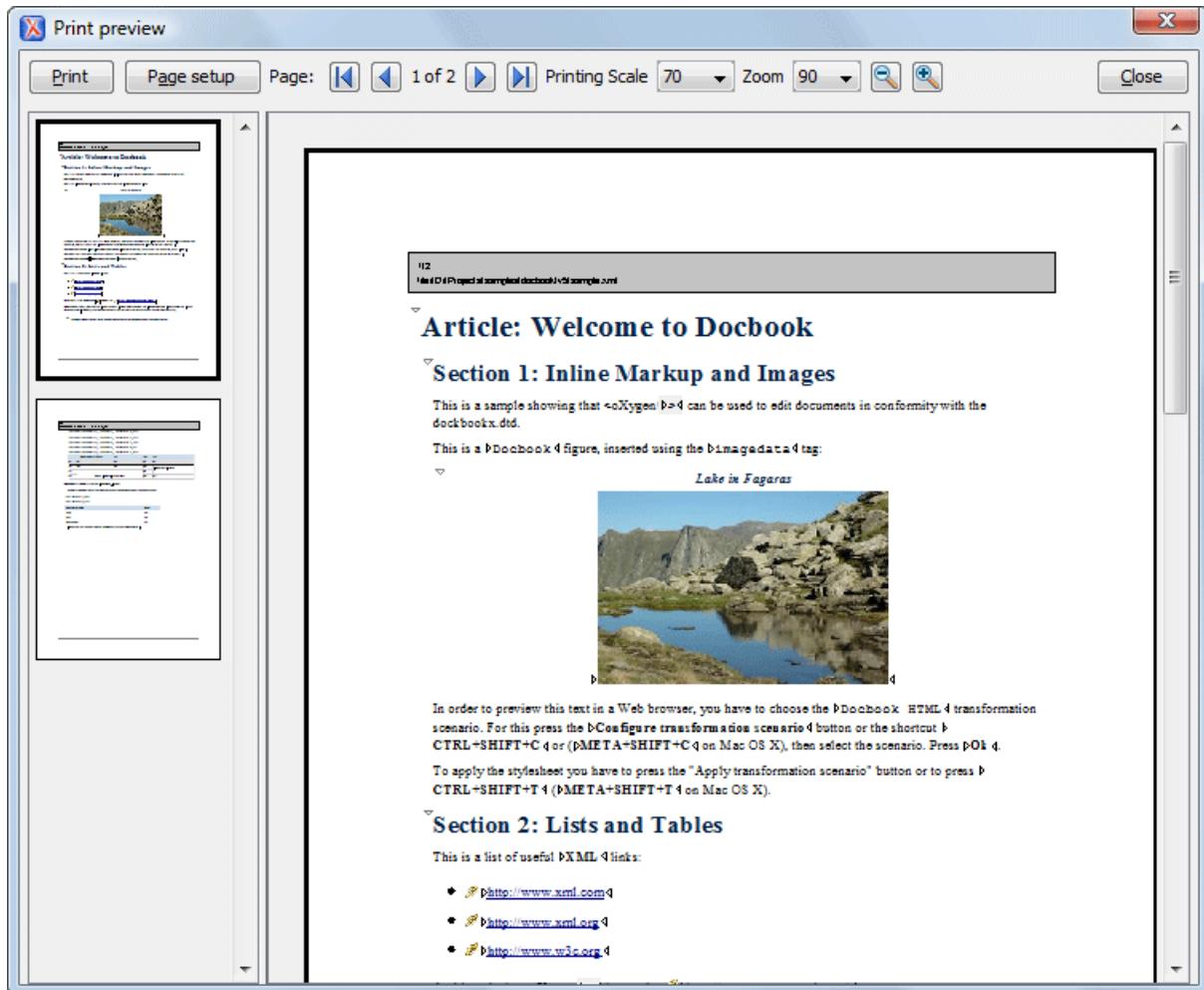


Figure 12: Print Preview Dialog

The main window is split in three sections:

- **Preview area** - Displays the formatted document page as it will appear on printed paper.
- **Left stripe** - The left-hand side stripe which displays a list of thumbnail pages. Clicking any of them displays the page content in the main preview area.
- **Toolbar** - The toolbar top area which contains controls for printing, page settings, page navigation, print scaling, and zoom.

 **Note:** If your document is open in Author mode and contains *Track Changes*, you can print (or print preview) a copy of the document as if all changes have been accepted by switching the *Track Changes Visualization Modes* to **View Final**.

Bidirectional Text Support in Text Mode

Bidirectional documents contain text in both directions, usually involving characters from unique types of alphabets.

If bidirectional text (such as Arabic or Hebrew languages), certain Asian languages (such as Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Sinhala, Thai, Khmer), or other special characters (such as combining characters) are detected in a document, Oxygen XML Editor displays a **Special Characters Detected** dialog box that prompts you to **Enable** or **Disable** support for these special characters.

You can also configure this support in the **Support for Special Characters** section of the **Open/Save** preferences page. To enable or disable this support, *open the Preferences dialog box* and go to **Editor > Open/Save**.



Note: Disabling this support may affect text rendering, cursor positioning and navigation, as well as text selection and management operations. If you need to open [very large documents](#), the bidirectional editing support can also be disabled to enhance performance while editing.



Restriction: Bidirectional content in the **Text** mode cannot be rendered using **Bold** or **Italic**.

Grid Editing Mode

To activate the **Grid** mode, select **Grid** at the bottom of the editing area. This type of editor displays the XML document as a structured grid of nested tables.

In case you are a non-technical user, you are able to modify the text content of the edited document without working with the XML tags directly. You can expand and collapse the tables using the mouse cursor and also display or hide the elements of the document as nested. The document structure can also be changed easily with drag and drop operations on the grid components. To zoom in and out, use **Ctrl + (Command + on OS X)**, **Ctrl - (Command - on OS X)**, **Ctrl 0 (Command 0 on OS X)** or **Ctrl Scroll Forward (Command Scroll Forward on OS X) / Ctrl Scroll Backwards (Command Scroll Backwards on OS X)**.

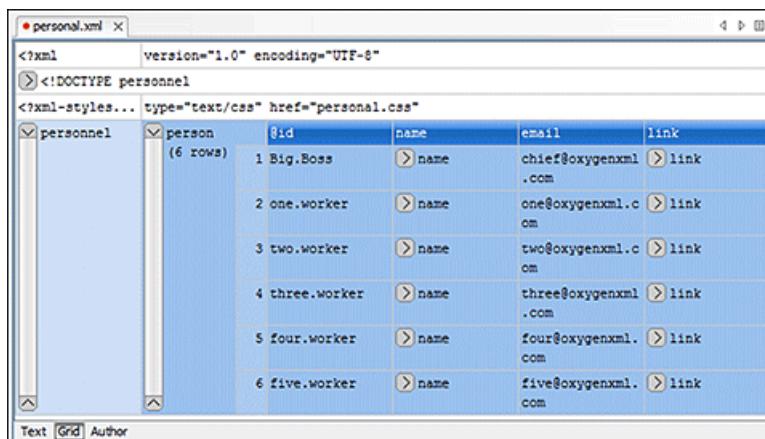


Figure 13: The Grid Editor

To switch back from the **Grid** mode to the **Text** or **Author** mode, use the **Text** and **Grid** buttons from the bottom of the editor. You are also able to perform this switch from **Document > Edit Mode > Grid** and **Document > Edit Mode > Text**.

If the edited document is associated with a schema (DTD, XML Schema, Relax NG, etc.), the editor offers **Content Completion Assistant** for the elements and attributes names and values. If you choose to insert an element that has required content, the sub-tree of needed elements and attributes are automatically included.

To display the content completion pop-up, you have to start editing (for example, double click a cell). Pressing **Ctrl Space** (**Command Space** on OS X) on your keyboard also displays the pop-up.

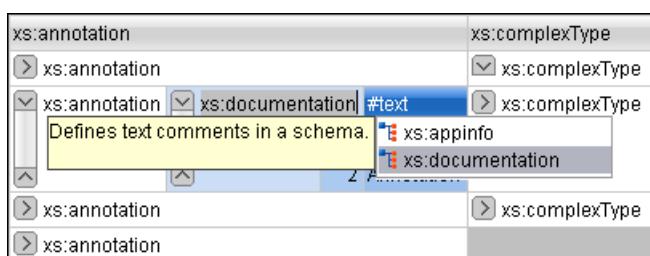


Figure 14: Content Completion in Grid Editor

To watch our video demonstration about some of the features available in the **Grid** editor, go to http://oxygengxml.com/demo/Grid_Editor.html.

Layouts: Grid and Tree

The **Grid** editor offers two layout modes. The default one is the grid layout. This smart layout detects the recurring elements in the XML document and creates tables having the children (including the attributes) of these elements as columns. This way, it is possible to have tables nested in other tables, reflecting the structure of your document.

<?xml version="1.0" encoding="UTF-8"					
test	table	tr (3 rows)	@id	first	last
			1	10001	Jhon
			2	10002	Mark
			3	10003	Dave
					Flint

Figure 15: Grid Layout

The other layout mode is tree-like. It does not create any tables and it only presents the structure of the document.

<?xml version="1.0" encoding="UTF-8"				
test	table	tr	@id	10001
			first	Jhon
			last	Doe
		tr	@id	10002
			first	Mark
			last	Ewing
		tr	@id	10003
			first	Dave
			last	Flint

Figure 16: Tree Layout

To switch between the two modes, go to **Document > Grid Layout > Grid mode/Tree mode**.

Grid Move Navigation

At first, the content of a document opened in the **Grid** mode is collapsed. Only the root element and its attributes are displayed. The grid disposition of the node names and values is similar to a web form or dialog box. The same set of key shortcuts used to select dialog box components is also available in the **Grid** mode:

Table 3: Shortcuts in the Grid Mode

Key	Action
<u>Tab</u>	Moves the caret to the next editable value in a table row.
<u>Shift Tab</u>	Moves the caret to the previous editable value in a table row.
<u>Enter</u>	Begins editing and lets you insert a new value. Also commits the changes after you finish editing.
<u>Up Arrow/Page Up</u>	Navigates toward the beginning of the document.
<u>Down Arrow/Page Down</u>	Navigates toward the end of the document.
<u>Shift</u>	Used in conjunction with the navigation keys to create a continuous selection area.

Key	Action
Ctrl (Command on OS X) key	Used in conjunction with the mouse cursor to create discontinuous selection areas.

The following key combinations can be used to scroll the grid:

- **Ctrl Up Arrow (Command Up Arrow on OS X)** - scrolls the grid upwards.
- **Ctrl Down Arrow (Command Down Arrow on OS X)** - scrolls the grid downwards.
- **Ctrl Left Arrow (Command Left Arrow on OS X)** scrolls the grid to the left.
- **Ctrl Right Arrow (Command Right Arrow on OS X)** scrolls the grid to the right.

An arrow sign displayed at the left of the node name indicates that this node has child nodes. To display the children, click this sign. The expand/collapse actions can be invoked either with the **NumPad+** and **NumPad-** keys, or from the **Expand/Collapse** submenu of the contextual menu or from **Document > Grid Expand/Collapse**.

The following actions are available on the **Expand/Collapse** menu:

 **Expand All**

Expands the selection and all its children.

 **Collapse All**

Collapses the selection and all its children.

Expand Children

Expands all the children of the selection but not the selection.

Collapse Children

Collapses all the children of the selection but not the selection.

Collapse Others

Collapses all the siblings of the current selection but not the selection.

Specific Grid Actions

In order to access these actions, you can click the column header and choose the **Table** item from the contextual menu. The same set of actions is available in the **Document** menu and on the **Grid** toolbar which is opened from menu **Window > Show Toolbar > Grid**.

Sorting a Table Column

You can sort certain table columns by using the  Sort ascending or  Sort descending actions that are available on the **Grid** toolbar or from the contextual menu.

The sorting result depends on the data type of the column content. It can be different in case of number (numerical sorting) or text information (alphabetical sorting). The editor automatically analyzes the content and decides what type of sorting to apply. When a mixed set of values is present in the sorted column, a dialog box is displayed that allows you to choose the desired type of sorting between *numerical* and *alphabetical*.

Inserting a Row in a Table

You can add a new row using the **Copy/Paste** actions, or by selecting  **Insert row** from the contextual menu or the **Grid** toolbar.

For a faster way to insert a new row, move the selection over the row header, and then press **Enter**. The row header is the zone in the left of the row that holds the row number. The new row is inserted below the selection.

Inserting a Column in a Table

You can insert a column after the selected column by using the  **Insert column** action from the contextual menu or the **Grid** toolbar.

Clearing the Content of a Column

You can clear all the cells from a column by using the **Clear content** action from the contextual menu.

Adding Nodes

You can add nodes before, after, or as last child of the currently selected node by using the various actions in the following submenus of the contextual menu:

- **Insert before**
- **Insert after**
- **Append child**

Duplicating Nodes

You can quickly create new nodes by duplicating existing ones. The **Duplicate** action is available in the contextual menu and in the **Document > Grid Edit** menu.

Refresh Layout

When using drag and drop to reorganize the document, the resulting layout can be different from the expected one. For instance, the layout can contain a set of sibling tables that can be joined together. To force the layout to be recomputed, you can use the  **Refresh selected** action that is available in the contextual menu and in the **Document > Grid Edit** menu.

Start and Stop Editing a Cell Value

To edit the value of a cell, simply select the grid cell and press **(Enter)** or you can use the  **Start Editing** action found in the **Document > Grid Edit** menu.

To stop editing a cell value, press **(Enter)** again or use the  **End Editing** action found in the **Document > Grid Edit** menu.

To cancel the editing without saving the current changes in the document, press the **(Esc)** key.

Drag and Drop in the Grid Editor

You are able to easily arrange different sections in your XML document in the **Grid** mode by using drag and drop actions.

You can do the following with drag and drop:

- Copy or move a set of nodes.
- Change the order of columns in the tables.
- Move the rows from the tables.

These operations are available for both single and multiple selections. To deselect one of the selected fragments, use **Ctrl Click (Command Click on OS X)**.

While dragging, the editor paints guide-lines showing the locations where you can drop the nodes. You can also drag nodes outside the **Grid** editor and text from other applications into the **Grid**. For more information, see [Copy and Paste in the Grid Editor](#).

Copy and Paste in the Grid Editor

The selection in the **Grid** mode is a bit complex compared to the selection in a text component. It consists of a current selected cell and additional selected cells. These additional cells are either hand picked by you with the cursor, or implied by the current selected cell. To be more specific, let's consider you click the name of the column - this becomes the current selected cell, but the editor automatically extends the selection so that it contains all the cells from that column. The current selected cell is painted with a color that is different from the rest of the selection.

You can select discontinuous regions of nodes and place them in the clipboard using the copy action. To deselect one of the selected fragments, use **Ctrl Click (Command Click on OS X)**. Pasting these nodes relative to the current selected

cell may be done in two ways: just below (after) as a brother, which is the default behavior, or as the last child of the selected cell.

The **Paste as Child** action is available in the contextual menu.

The same action can be found in the menu **Document > Grid Edit > Paste as Child**.

The nodes copied from the **Grid** editor can also be pasted into the **Text** editor or other applications. When copying from the **Grid** into the **Text** editor or other text based applications, the inserted string represents the nodes serialization. The nodes from tables can be copied using HTML or RTF in table format. The resulting cells contain only the concatenated values of the text nodes.

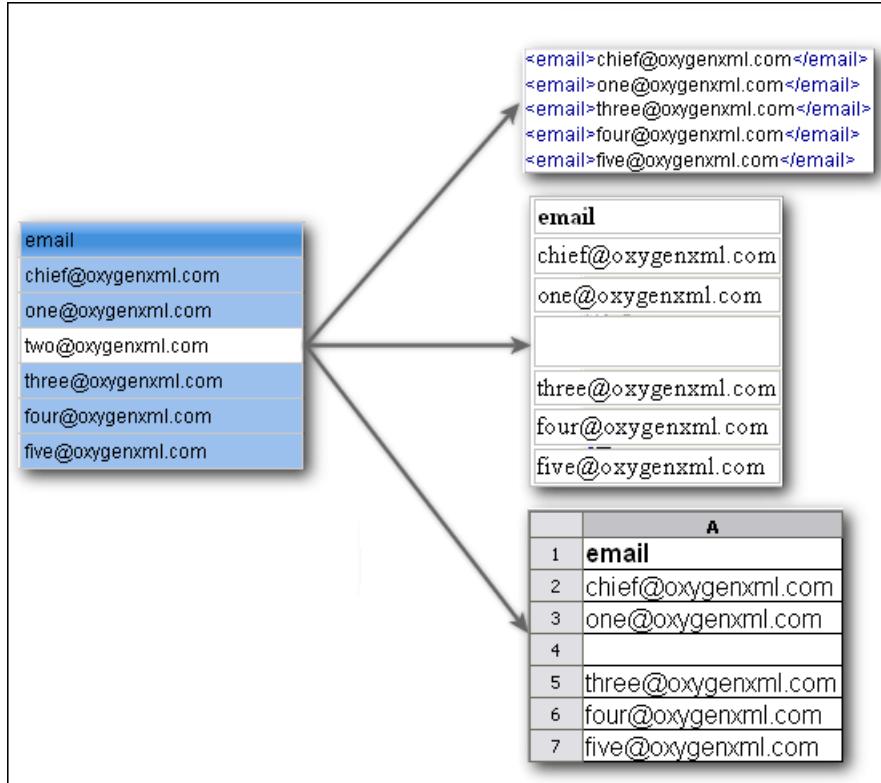


Figure 17: Copying from Grid to Other Editors

In the **Grid** editor you can paste well-formed XML content or tab separated values from other editors. If you paste XML content, the result will be the insertion of the nodes obtained by parsing this content.

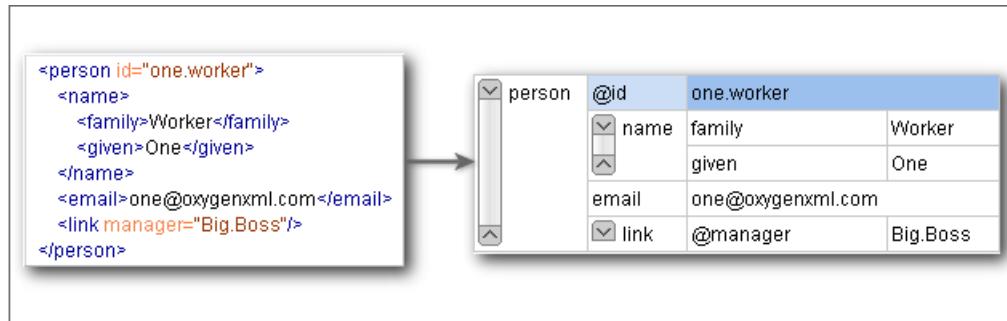


Figure 18: Copying XML Data into Grid

If the pasted text contains multiple lines of tab separated values it can be considered as a matrix of values. By pasting this matrix of values into the **Grid** editor the result will be a matrix of cells. If the operation is performed inside existing cells, the existing values will be overwritten and new cells will be created when needed. This is useful, for example, when trying to transfer data from Excel like editors into the **Grid** editor.

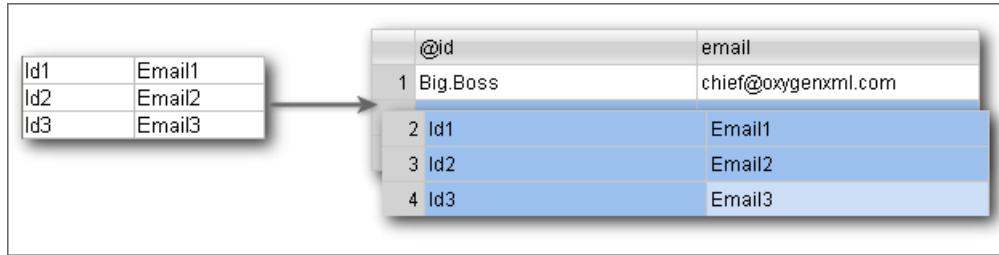


Figure 19: Copying Tab Separated Values into Grid

Bidirectional Text Support in Grid Mode

If you are editing documents employing a different text orientation, you can change the way the text is rendered and edited in the grid cells by using the **Ctrl Shift O (Command Shift O on OS X)** shortcut to switch from the default left to right text orientation to the right to left orientation.



Note: This change applies only to the text from the cells, and not to the layout of the grid editor.

<?xml	version="1.0" encoding="UTF-8"
sample (9 rows)	#text
	عندما يريد العالم أن يتكلّم، فهو ينحني بلغة بونيكود.
1	Quan el món vol conversar, parla Unicode
2	כארה העולם רוצה לדבר, הוא מדבר ב-Unicode
3	Ha a világ beszélni akar, azt Unicode-ul mondja
4	Quando il mondo vuole comunicare, parla Unicode
5	世界的に話すなら、Unicode です。
6	세계를 할한 대화, 유니코드로 하십시오
7	Når verden vil snakke, snakker den Unicode
8	Når verda ønskjer å snakke, talar ho Unicode
9	

Figure 20: Default left to right text orientation

<?xml	"version="1.0" encoding="UTF-8"
sample (9 rows)	#text
1	عندما يريد العالم أن يتكلّم، فهو ينحني بلغة بونيكود.
2	Quan el món vol conversar, parla Unicode
3	כארה העולם רוצה לדבר, הוא מדבר ב-Unicode
4	Ha a világ beszélni akar, azt Unicode-ul mondja
5	Quando il mondo vuole comunicare, parla Unicode
6	。世界的に話すなら、Unicode です
7	세계를 할한 대화, 유니코드로 하십시오
8	Når verden vil snakke, snakker den Unicode
9	Når verda ønskjer å snakke, talar ho Unicode

Figure 21: Right to left text orientation

Author Editing Mode

This chapter presents the WYSIWYG-like visual editor, called **Author** mode, that is targeted to content authors.

Tagless XML Authoring

Once the structure of an XML document and the required restrictions on its elements and their attributes are defined with an XML schema, the editing of the document becomes easier in a WYSIWYG-style editor in which the XML markup is not visible.

This type of tagless editor is available in Oxygen XML Editor as the **Author** mode. To enter this mode, click the **Author** button at the bottom of the editing area. The **Author** mode renders the content of the XML document visually, based on a CSS stylesheet associated with the document. Many of the actions and features available in **Text** mode are also available in **Author** mode.

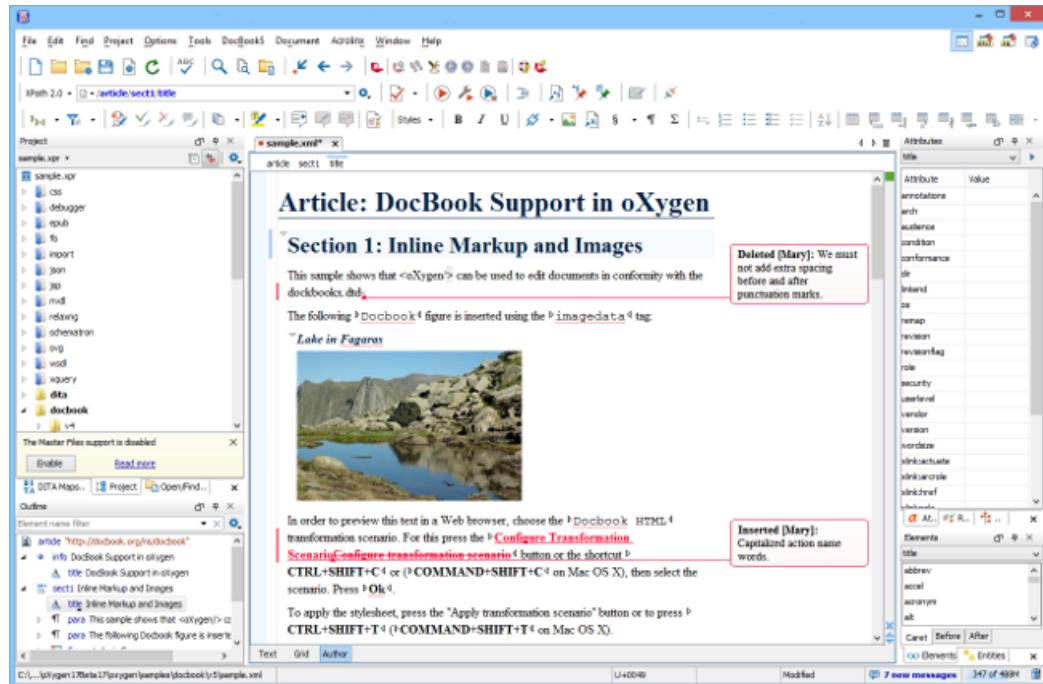


Figure 22: Author Editing Mode

Associating a Stylesheet with an XML Document

The tagless rendering of an XML document in the **Author** mode is driven by a CSS stylesheet which conforms to the [version 2.1 of the CSS specification](#) from the W3C consortium. Some CSS 3 features, such as namespaces and custom extensions, of the CSS specification are also supported. Oxygen XML Editor also supports stylesheets coded with the LESS dynamic stylesheet language.

There are several methods for associating a stylesheet (CSS or LESS) with an XML document:

1. Insert the `<xml-stylesheet` processing instruction with the `type` attribute at the beginning of the XML document. If you do not want to alter your XML documents, *you should set-up a document type*.

CSS example:

```
<?xml-stylesheet type="text/css" href="test.css"?>
```

LESS example:

```
<?xml-stylesheet type="text/css" href="test.less"?>
```



Note: XHTML documents need a `link` element, with the `href` and `type` attributes in the `head` child element, as specified in the [W3C CSS specification](#). XHTML example:

```
<link href="/style/screen.css" rel="stylesheet" type="text/css"/>
```

2. Configure a *Document Type Association* by adding a new CSS or LESS file in the settings. To do so, [open the Preferences dialog box](#) and go to **Document Type Association**. Edit the appropriate framework, open the **Author** tab, then the **CSS** tab. Press the  New button to add a new CSS or LESS file.



Note: The Document Type Associations are read-only, so you need to extend an existing one.

Selecting and Combining Multiple CSS Styles

Oxygen XML Editor provides a **Styles** drop-down list on the **Author Styles** toolbar that allows you to select one main (*non-alternate*) CSS style and multiple *alternate* CSS styles. An option in the preferences can be enabled to allow the alternate styles to behave like layers and be combined with the main CSS style. This makes it easy to change the look of the document.

An example of a common use case is when content authors want to use custom styling within a document. You can select a main CSS stylesheet that styles the whole document and then apply alternate styles, as layers, to specific parts of the document. In the subsequent figure, a DITA document has the **Century** style selected for the main CSS and the alternate styles **Full width**, **Show table column specification**, **Hints**, and **Inline actions** are combined for additive styling to specific parts of the document.



Note: Oxygen XML Editor comes with a set of predefined CSS layer stylesheets for DITA documents only, but the support is available for all other document types.



Tip: The **Hints** style displays tooltips throughout DITA documents that offer additional information to help you with the DITA structure. The **Inline actions** style displays possible elements that are allowed to be inserted at various locations throughout DITA documents.

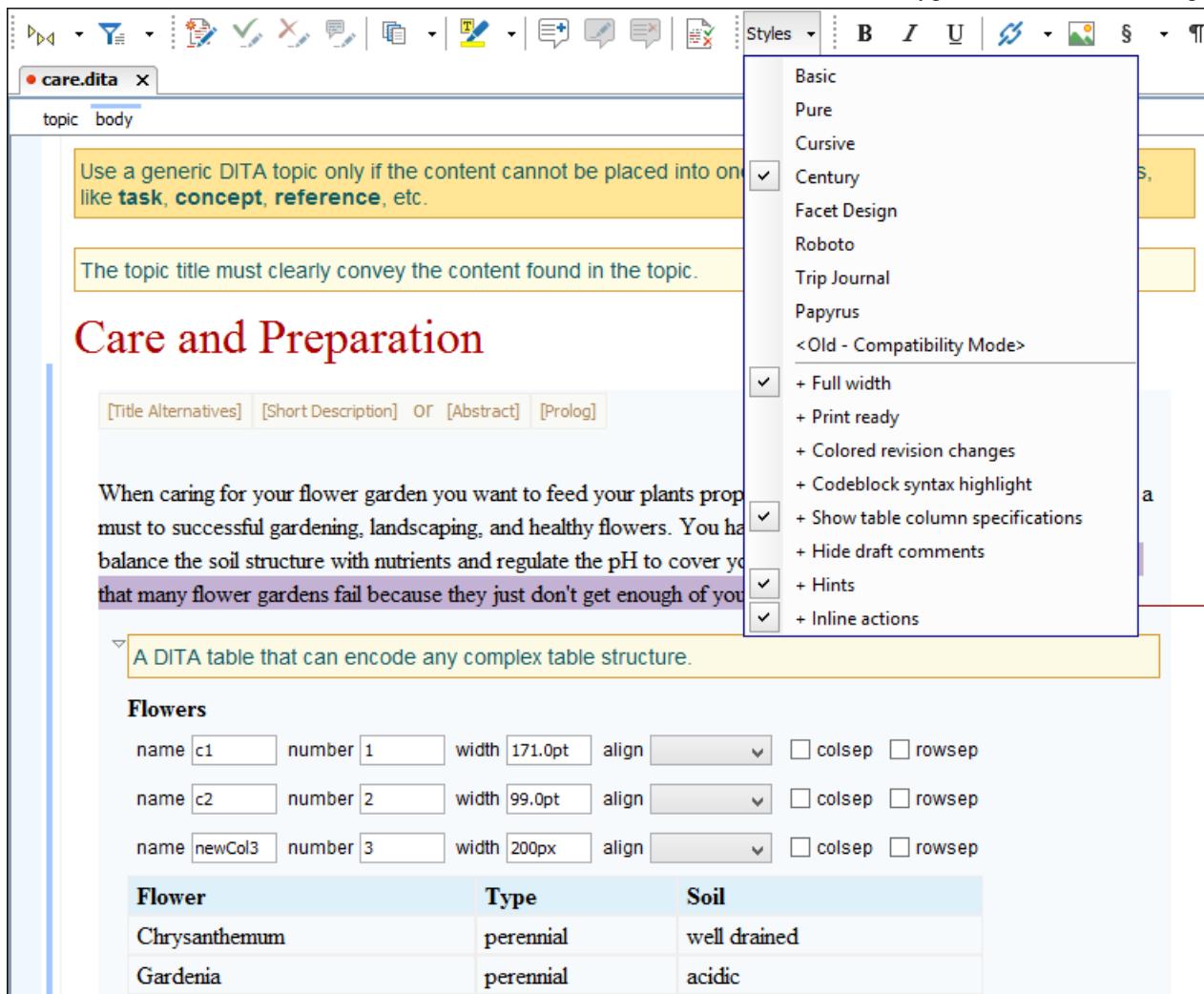


Figure 23: Styles Drop-down List in a DITA Document

Author Mode User Roles

There are two main types of users of the **Author** mode: *framework developers* and *content authors*. A *framework developer* is a technical person with advanced XML knowledge who defines the framework for authoring XML documents in the tagless editor. Once the framework is created or edited by the developer, it is distributed as a deliverable component ready to plug into the application for the content authors. A *content author* does not need to have advanced knowledge about XML tags, operations such as validation of XML documents, or applying an XPath expression to an XML document. The *content author* just uses the framework set-up by the developer in the application and starts editing the content of XML documents without editing the XML tags directly.

The framework set-up by the developer is also called *document type association* and defines a type of XML document by specifying all the details needed for editing the content of XML documents in tagless mode.

The framework details that are created and customized by the developer include:

- the CSS stylesheet that drives the tagless visual rendering of the document
- the rules for associating an XML schema with the document, which is needed for content completion and validation of the document
- transformation scenarios for the document
- XML catalogs
- custom actions available as buttons on the toolbar

The tagless editor comes with some ready-to-use predefined document types for XML frameworks such as DocBook, DITA, TEI, and XHTML.

To watch our video demonstration about the basic functionality of the **Author** mode, go to http://oxygentools.com/demo/WYSIWYG_XML_Editing.html.

General Author Presentation

A content author edits the content of XML documents in the **Author** mode disregarding the XML tags as they are not visible in the editor. If he edits documents conforming to one of the predefined types he does not need to configure anything as the predefined document types are already configured when the application is installed. Otherwise he must plug the configuration of the document type into the application. This is as easy as unzipping an archive directly in the [OXYGEN_DIR] / frameworks folder.

In case the edited XML document does not belong to one of the document types *set up in Preferences* you can specify the CSS stylesheets to be used by inserting an `<xmlstylesheet type="text/css" media="media type" title="title" href="URL" alternate="yes|no"?>`

The syntax of such a processing instruction is:

```
<xmlstylesheet type="text/css" media="media type" title="title"
href="URL" alternate="yes|no"?>
```

You can read more about associating a CSS to a document in the section about [customizing the CSS of a document type](#).

When the document has no CSS association or the referenced stylesheet files cannot be loaded, a default one is used. A warning message is also displayed at the beginning of the document presenting the reason why the CSS cannot be loaded.

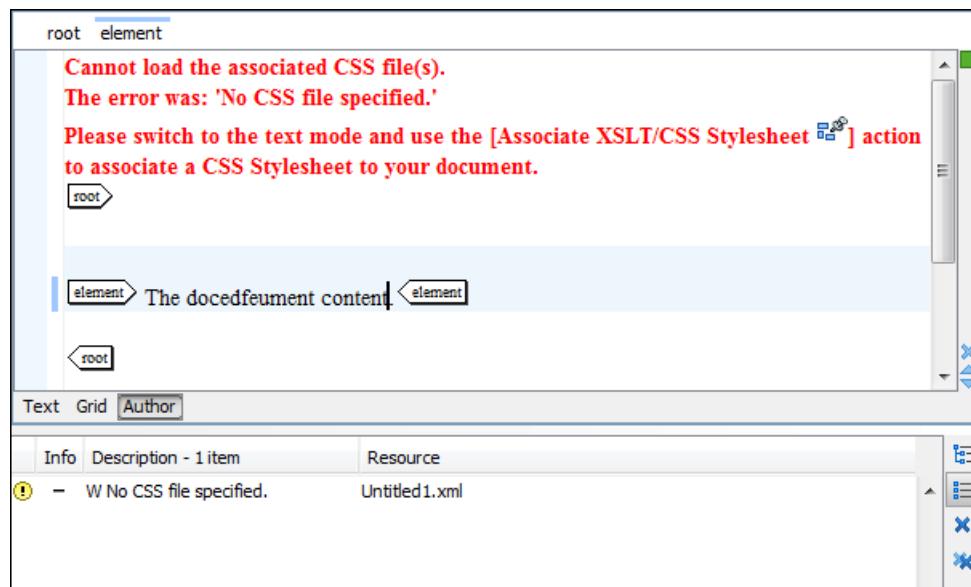


Figure 24: Document with no CSS association default rendering

Author Views

The content author is supported by special views which are automatically synchronized with the current editing context of the editor panel. The views present additional information about this context thus helping the author to see quickly the current location in the overall document structure and the available editing options.

Outline View

The **Outline** view offers the following functionality:

- [Document Overview](#)
- [Outline View Specific Actions](#)

- [Modification Follow-up](#)
- [Document Structure Change](#)
- [Document Tag Selection](#)

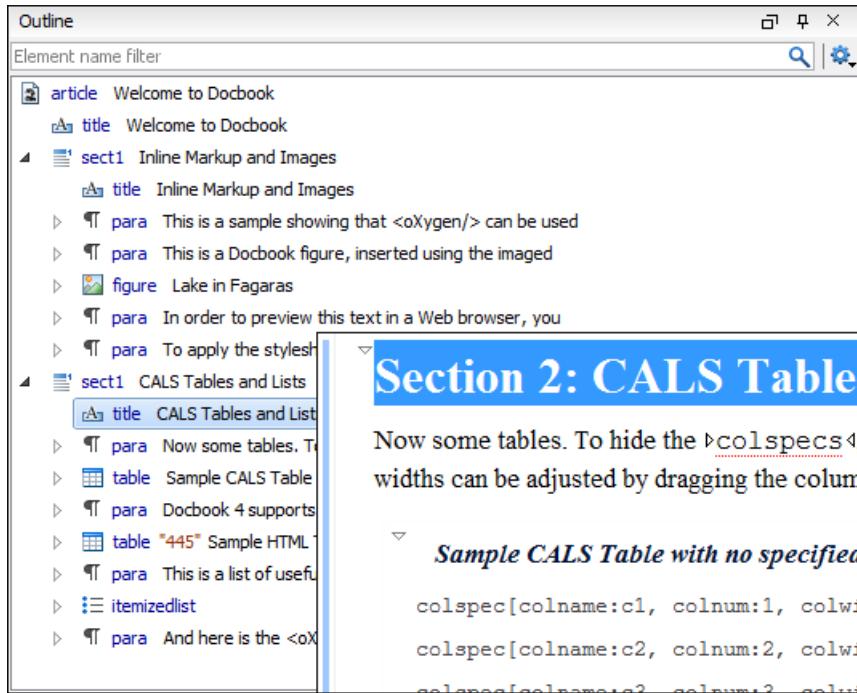


Figure 25: The Outline View

XML Document Overview

The **Outline** view displays a general tag overview of the current edited XML document. It also shows the correct hierarchical dependencies between the tag elements. This functionality makes it easier for the user to be aware of the document structure and the way tags are nested.

The **Outline** view allows you to:

- Insert or delete nodes using pop-up menu actions.
- Move elements by dragging them to a new position in the tree structure.
- Highlight elements in the **Author** editor area.



Note: The **Outline** view is synchronized with the **Author** editor area. When you make a selection in the **Author** editor area, the corresponding elements of the selection are highlighted in the **Outline** view and vice versa. This functionality is available both for single and multiple selection. To deselect one of the elements, use [Ctrl Click \(Command Click on OS X\)](#).

Document errors (such as an element inserted in an invalid position, or a wrong attribute name, or a missing required attribute value) are highlighted in the **Outline** tree:

- A red exclamation mark decorates the element icon.
- A dotted red underline decorates the element name and value.
- A tooltip provides more information about the nature of the error, when you hover with the mouse pointer over the faulted element.

Modification Follow-up

When you edit a document, the **Outline** view dynamically follows the changes that you make, displaying the node that you modify in the middle of the view. This functionality gives you great insight on the location of your modifications in the document that you edit.

Document Structure Change

Entire XML elements can be moved or copied in the edited document using only the mouse in the **Outline** view in drag-and-drop operations. Several drag and drop actions are possible:

- If you drag an XML element in the **Outline** view and drop it on another one in the same panel then the dragged element will be moved after the drop target element.
- If you hold the mouse pointer over the drop target for a short time before the drop then the drop target element will be expanded first and the dragged element will be moved inside the drop target element after its opening tag.
- You can also drop an element before or after another element if you hold the mouse pointer towards the upper or lower part of the targeted element. A marker will indicate whether the drop will be performed before or after the target element.
- If you hold down the (**Ctrl (Command on OS X)**) key after dragging, there will be performed a copy operation instead of a move one.

The drag and drop action in the **Outline** view can be *disabled and enabled from the Preferences dialog*.

 **Tip:** You can select and drag multiple nodes in the Author Outline tree.

Outline Filters

The following actions are available in the  **Settings** menu on the Outline view's toolbar:

 **Flat presentation mode of the filtered results**

When active, the application flattens the filtered result elements to a single level.

 **Show comments and processing instructions**

Show/hide comments and processing instructions in the **Outline** view.

 **Show element name**

Show/hide element name.

 **Show text**

Show/hide additional text content for the displayed elements.

 **Show attributes**

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from *the Outline preferences panel*.

 **Configure displayed attributes**

Displays the *XML Structured Outline preferences page*.

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

The Contextual Menu of the Outline Tree

The contextual menu of the **Outline** tree contains the following actions:

Edit attributes

Opens a dialog box that allow you to see and edit the attributes of the selected node.

Append child, Insert before, and Insert after

Submenus that allow you to quickly insert new tags in the document at the place of the element selected in the **Outline** tree. The **Append child** submenu lists the names of all the elements which are allowed by the schema associated with the current document as child of the current element. The effect is the same as typing the '<' character and selecting an element name from the popup menu offered by *the Content Completion Assistant*. The **Insert before** and **Insert after** submenus list the elements which are allowed by the schema associated with the current document as siblings of the current element inserted immediately before respectively after the current element.

Cut, Copy, and Paste

Usual text manipulation actions. The content retains text attributes like color and font.

Paste

Pastes the clipboard content in the currently selected node. The **Paste Before** and **Paste After** variants of the *Paste* action allow you to paste the clipboard content before and after the currently selected node.

Delete

Deletes the currently selected node.

Toggle Comment

If the currently selected element is not commented, this action encloses it in an XML comment. Otherwise, it removes the comment.

Rename Element

Allows you to rename the selected element. Alternatively, you can choose to rename all its siblings with the same name or all elements with the same name in the entire document.

Expand More / Collapse All

Expand / collapse the selection and all its children.

-  **Tip:** You can copy, cut or delete multiple nodes in the **Outline** by using the contextual menu after selecting multiple nodes in the tree.

Elements View

The **Elements** view presents a list of all defined elements that you can insert in your document. All elements from a sequence are presented but the invalid proposals (which cannot be inserted in the current context) are grayed-out. The upper part of the view features a combo box that contains the current element's ordered ancestors. Selecting a new element in this combo box updates the list of the allowed elements in **Before** and **After** tabs.

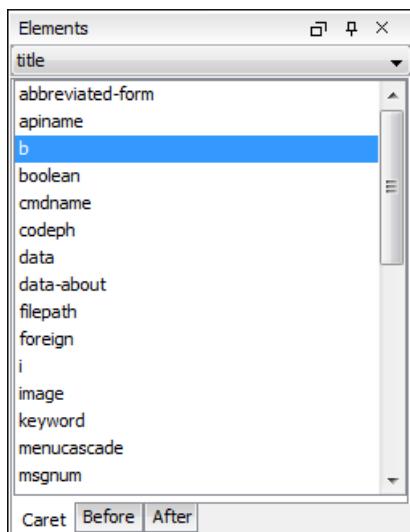


Figure 26: The Elements View

Three tabs present information relative to the caret location:

- **Caret** - Shows a list of all the elements allowed at the current caret location. Double-clicking any of the listed elements inserts that element at the caret position.
- **Before** - Shows a list of all elements that can be inserted before the element selected in the combo box. Double-clicking any of the listed elements inserts that element before the element at the caret position.
- **After** - Shows a list of all elements that can be inserted after the element selected in the combo box. Double-clicking any of the listed elements inserts that element after the element at the caret position.

Double clicking an element name in the list surrounds the current selection in the editor panel with the start tags and end tags of the element. If there is no selection, just an empty element is inserted in the editor panel at the cursor position.

Attributes View

The **Attributes** view presents all the attributes of the current element determined by the schema of the document. It allows you to insert attributes in the current element or change the value of the attributes already inserted. The attributes are rendered differently depending on their state:

- The names of the attributes with a specified value are rendered with a bold font, and their value with a plain font.
-  **Note:** The names of the attributes with an empty string value are also rendered bold.
- Default values are rendered with a plain font, painted gray.
 - Empty values display the text "[empty]", painted gray.
 - Invalid attributes and values are painted red.

Double-click a cell in the **Value** column to edit the value of the corresponding attribute. In case the possible values of the attribute are specified as list in the schema of the edited document, the **Value** column acts as a combo box that allows you to insert the values in the document.

You can sort the attributes table by clicking the **Attribute** column header. The table contents can be sorted as follows:

- By attribute name in ascending order.
- By attribute name in descending order.
- Custom order, where the used attributes are displayed at the beginning of the table sorted in ascending order, followed by the rest of the allowed elements sorted in ascending order.

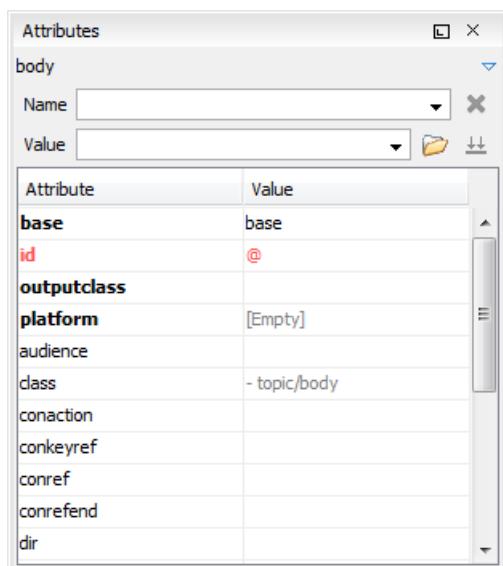


Figure 27: The Attributes View

A combo box located in the upper part of the view allows you to edit the attributes of the ancestors of the current element.

The following actions are available in the contextual menu:

Add

Allows you to insert a new attribute. Adding an attribute that is not in the list of all defined attributes is not possible when the *Allow only insertion of valid elements and attributes* schema aware option is enabled.

Set empty value

Specifies the current attribute value as empty.

Remove

Removes the attribute (action available only if the attribute is specified). You can invoke this action by pressing the **(Delete)** or **(Backspace)** keys.

Copy

Copies the `attrName="attrValue"` pair to the clipboard. The `attrValue` can be:

- The value of the attribute.
- The value of the default attribute, if the attribute does not appear in the edited document.
- Empty, if the attribute does not appear in the edited document and has no default value set.

Paste

This action is available in the contextual menu of the **Attributes** view, in the **Text** and **Author** modes. Depending on the content of the clipboard, the following cases are possible:

- If the clipboard contains an attribute and its value, both of them are introduced in the **Attributes** view. The attribute is selected and its value is changed if they exist in the **Attributes** view.
- If the clipboard contains an attribute name with an empty value, the attribute is introduced in the **Attributes** view and you can start editing it. The attribute is selected and you can start editing it if it exists in the **Attributes** view.
- If the clipboard only contains text, the value of the selected attribute is modified.

In-place Attributes Editor

To edit in-place the attributes of an XML element, do one of the following:

- Completely select the element or place the caret inside it and then press the **Alt Enter** keyboard shortcut.
- Double-click any named start tag when the document is edited in one of the following *display modes*: **Full Tags** with **Attributes**, **Full Tags**, **Block Tags**, **Inline Tags**.

This shortcut pops up a small window with the same content as the **Attributes** view. The default form of the pop-up window presents the **Name** and **Value** fields, with the list of all the possible attributes collapsed.

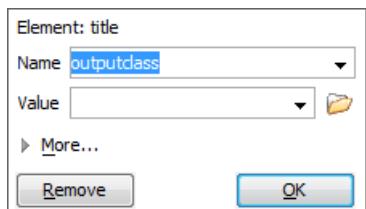


Figure 28: Edit attributes in place

The small right arrow button expands the list of possible attributes allowed by the schema of the document as in the **Attributes** view.

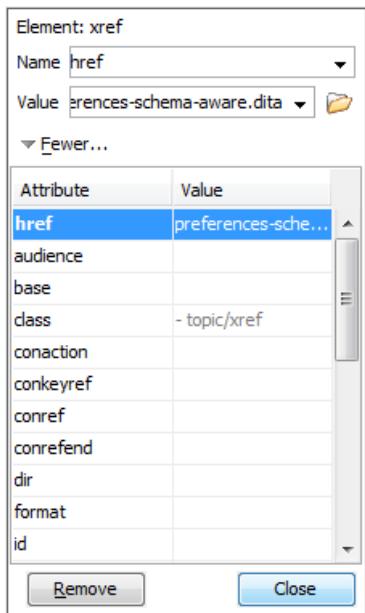


Figure 29: Edit attributes in place - full version

The **Name** field auto-completes the name of the attribute: the complete name of the attribute is suggested based on the prefix already typed in the field as the user types in the field.

Entities View

This view displays a list with all entities declared in the current document as well as built-in ones. Double clicking one of the entities will insert it at the current cursor position.

Entities	
Name	Value
lt	<
gt	>
amp	&
apos	'
quot	"
abbrev-d-att	(topic abbrev-d)
concept-att	(topic concept)
hazard-d-att	(topic hazard-d)
hi-d-att	(topic hi-d)
included-domains	&concept-att; ...
indexing-d-att	(topic indexing-d)
nbsps	
pr-d-att	(topic pr-d)
sw-d-att	(topic sw-d)
ui-d-att	(topic ui-d)
ut-d-att	(topic ut-d)

Figure 30: The Entities View

The view features a filtering capability that allows you to search an entity by name, value, or both. Also, you can choose to display the internal or external entities.

 **Note:** When entering filters, you can use the ? and * wildcards. Also, you can enter multiple filters by separating them with comma.

The Review View

The **Review** view is a framework-independent panel, available both for built-in, and custom XML document frameworks. It is designed to offer an enhanced way of monitoring all the changes that you make to a document. This means you are able to view and control highlighted, commented, inserted, and deleted content, or even changes made to attributes, using a single view.

The **Review** view is useful when you are working with documents that contain large quantities of edits. The edits are presented in a compact form, in the order they appear in the document. Each edit is marked with a type-specific icon.

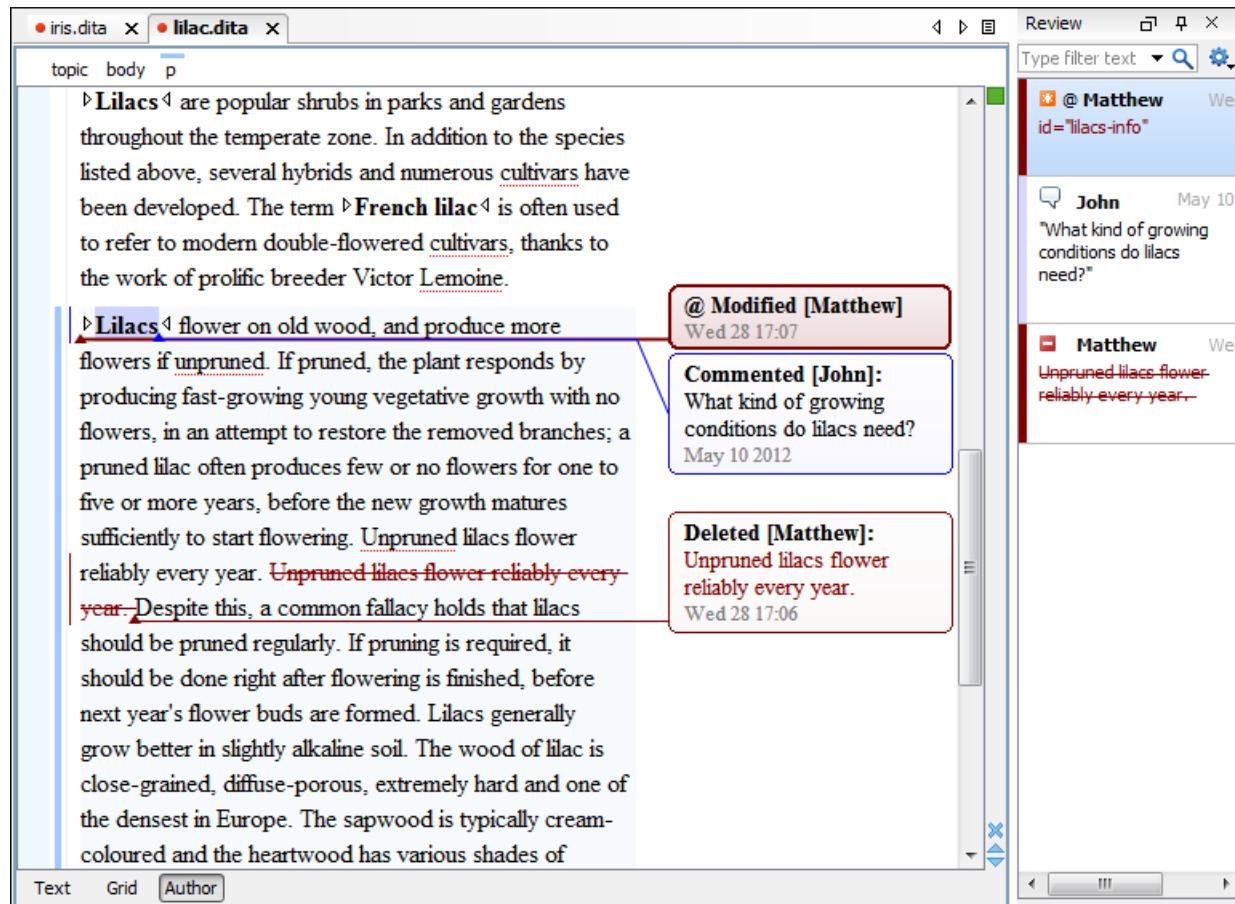


Figure 31: The Review View

To activate the **Review** view, do one of the following:

- click the **Manage reviews** button on the **Review** toolbar
- right click in a document and from the contextual menu go to **Review, Manage reviews**
- go to **Window > Show View > Review**

This view and the editing area are synchronized. When you select an edit listed in the **Review** view, its corresponding fragment of text is highlighted in the editing area and the reverse is also true. For example, when you place the caret inside an area of text marked as inserted, its corresponding edit is selected in the list.

The upper part of the view contains a filtering area which allows you to search for specific edits. Use the small arrow symbol from the right side of the search field to display the search history. The **Settings** button allows you to:

- **Show highlights** - controls whether the **Review** view displays the highlighting in your document.
- **Show comments** - controls whether the **Review** view displays the comments in the document you are editing.
- **Show track changes** - controls whether the **Review** view displays the inserted and deleted content in your document.
- **Show review time** - displays the time when the edits from the **Review** view were made.

The following actions are available when you hover the edits in the **Review** view, using the cursor:

Remove

Action available for highlights and comments presented in the **Review** view. Use this action to remove these highlights or comments from your document;

Accept

Action available for inserted and deleted content presented in the **Review** view. Use this action to accept the changes in your document;

Reject

Action available for inserted and deleted content presented in the **Review** view. Use this action to reject the changes in your document.

Depending on the type of an edit, the following actions are available in its contextual menu in the **Review** view:

Show comment

This option is available in the contextual menu of changes not made by you and of any comment listed in the **Review** view. Use this option to view a comment in the **Show comment** dialog.

Edit comment

This option is available in the contextual menu of your comments, listed in the **Review** view. Use this action to start editing the comment.

Remove comment

This option is available in the contextual menu of a comment listed in the **Review** view. Use this action to remove the selected comment.

Show only reviews by

This option is available in the contextual menu of any edit listed in the **Review** view. Use this action to keep visible only the edits of a certain author in the view.

Remove all comments

This option is available in the contextual menu of any comment listed in the **Review** view. Use this action to remove all the comments that appear in the edited document.

Change color

Opens a palette that allows you to choose a new color for the highlighted content.

Remove highlight

Removes the selected highlighting.

Remove highlights with the same color

Removes all the highlighting with the same color from the entire document.

Remove all highlights

Clears all the highlighting in your document.

Accept change

Accepts the selected change.

Reject change

Rejects the selected change.

Comment change

This option is available in the contextual menu of an insertion or deletion that you made. Use this option to open the **Edit comment** dialog and comment the change you made.

Accept all changes

Accepts all the changes made to a document.

Reject all changes

Rejects all the changes made to a document.

To watch our video demonstration about the **Review** view, go to http://oxygenxml.com/demo/Review_Panel.html.

CSS Inspector View

The purpose of the **CSS Inspector** view is to display information about the styles applied to the currently selected element.

You can use this view to examine the structure and layout of the CSS rules that match the element. The matching rules displayed in this view include a link to the line in the CSS file that defines the styles. With this tool you can see how the CSS rules were applied and the properties defined, and use the link to open the associated CSS for editing purposes.

```

CSS Inspector
test.css:44
*[attr='val'] {
    text-decoration:underline;
}

test.css:35
job {
    display:block;
    color:blue;
    margin-top:2em;
    margin-right:2em;
    margin-left:2em;
    margin-bottom:2em;
    font-weight:bold;
    font-size:large;
    text-decoration:none;
}

Inherited from doc
test.css:1
doc {
    font-family:arial , helvetica , sans-serif;
}

Element :before :after Computed Path

```

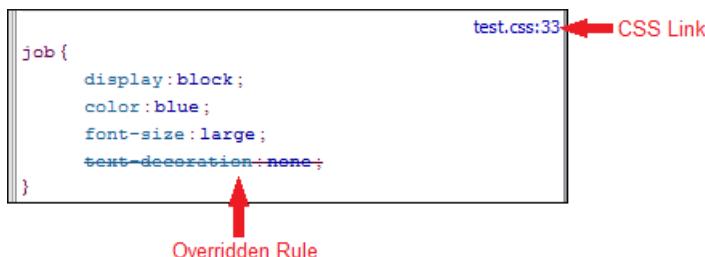
Figure 32: CSS Inspector View

Displaying the CSS Inspector View

You can open this view by selecting the **Inspect Styles** action from the contextual menu in **Author** mode, or selecting the **CSS Inspector** view in the **Window > Show View** menu. This action makes the view visible and also initializes it for the currently selected element.

Displaying Rules

All rules that apply to the current element are displayed in sections, which are listed in order of importance (from most specific to least specific). Rules that are overridden by other rules are crossed out. If you click on the link in the top-right corner of a rule Oxygen XML Editor opens the associated CSS file at the line number where the properties of the rule are defined.



The **CSS Inspector** view contains five tabs:

- **Element** - displays the CSS rules matching the currently selected element in the **Author** page (ordered from most-specific to least-specific)
- **:before** - displays the rules matching the **:before** pseudo-element
- **:after** - displays the rules matching the **:after** pseudo-element
- **Computed** - displays all the styling properties that apply to the current element, as a result of all the CSS rules matching the element
- **Path** - displays the path for the current element, and its attributes, allowing you to quickly see the attributes on all parent elements, and allows you to copy fragments from this view and paste it into the associated CSS to easily create new rules

The information displayed in each of the five tabs is updated when you click on different elements in the **Author** editing view. The first three tabs include the link to the associated CSS source, while the other two tabs simply display the style properties that match the current element.

Each of the tabbed panes include a contextual menu with the following actions:

- **Copy** - copies the current selection
- **Select all** - selects all information listed in the pane
- **Show empty rules** - forces the view to show all the matching rules, even if they do not declare any CSS properties (by default, the empty rules are not displayed)

The Author Editor

This section explains the features of the tag-less WYSIWYG-like editor for XML documents.

Navigating the Document Content

Using the Keyboard

Oxygen XML Editor allows you to quickly navigate through a document using **Tab** to go to the next XML node and **Shift Tab** to go to the previous one. The caret is moved to the next / previous editable position. When the caret is positioned in a space preserve element, press a key on your keyboard and then use **Tab** to arrange the text. You can also arrange the text using **Tab** if you position the cursor in a space preserve element using your mouse. In case you encounter a space preserve element when you navigate through a document and you press no other key, the next **Tab** continues the navigation.

To navigate one word forward or backwards, use **Ctrl Right Arrow (Command Right Arrow on OS X)**, and **Ctrl Left Arrow (Command Left Arrow on OS X)**, respectively. Entities and hidden elements are skipped. To position the cursor at the beginning or at the end of the document you can use **Ctrl Home (Command Home on OS X)**, and **Ctrl End (Command End on OS X)** respectively.

Using the Navigation Toolbar

The locations of selected text are stored in an internal list which allows you to navigate between them with the  **Back** (**Ctrl Alt [(Command Alt [on OS X)**) and  **Forward (Ctrl Alt] (Command Alt] on OS X)**) buttons from the **Navigation** toolbar. The  **Last Modification (Ctrl Alt G (Command Alt G on OS X))** button automatically takes you to the latest edited text.

Using the Breadcrumb Helpers

A top stripe called *breadcrumb* indicates the path from document root to the current element.

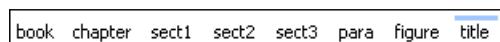


Figure 33: The breadcrumb in Editor view

The last element is also highlighted by a thin light blue bar for easier identification. Clicking one element from the top stripe selects the entire element in the editor view. Also, each element provides a contextual popup menu with access to the following actions:

- The **Edit Attributes** action which opens the *in-place attributes editor*.
- The **Edit Profiling Attributes** action allows you to select the *profiling attributes* that apply to a certain element.
- The **Append child**, **Insert before** and **Insert after** submenus of the popup menu allow you to insert new tags in the document at the place of the selected element. The **Append child** submenu lists the names of all the elements which are allowed by the schema associated with the current document as child of the current element. The effect is the same as selecting an element name from the popup menu offered by *the content completion assistant*. The **Insert before** and **Insert after** submenus list the elements which are allowed by the schema associated with the current document as siblings of the current element inserted immediately before respectively after the current element.
- The **Cut**, **Copy**, **Paste** and **Delete** items of the popup menu execute *the same actions as the Edit menu items with the same name* on the elements currently selected in the stripe. The **Cut** and **Copy** operations (like the `display:block` property or the tabular format of the data from a set of table cells) preserve the styles of the copied content. The **Paste before**, **Paste after** and **Paste as Child** actions allow the user to insert an well-formed element before, after or as a child of the currently selected element.
- The **Toggle Comment** item of the **Outline** tree popup menu encloses the currently selected element of the top stripe in an XML comment, if the element is not commented, or removes the comment if it is commented.
- Using the **Rename Element** action the selected element and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog box.

 **Tip:** The tag names displayed in the breadcrumb can be customized with an Author extension class that implements `AuthorBreadCrumbCustomizer`. See the [Oxygen SDK](#) for more details.

Using the Folding Support

When working on a large document, the **folding support** can be used to collapse some elements content leaving in focus only the ones you need to edit. Foldable elements are marked with a small triangle painted in the upper left corner. Hovering with the mouse pointer over that marker, the entire content of the element is highlighted by a dotted border for quick identification of the foldable area. The following actions are available in the contextual menu, **Folding** sub-menu:

- ▶ **Toggle Fold**
Toggles the state of the current fold.
- ▶ **Close Other Folds (Ctrl NumPad/ (Command NumPad/ on OS X))**
Folds all the elements except the current element.
- ▶ **Collapse Child Folds (Ctrl NumPad. (Command NumPad. on OS X))**
Folds the elements indented with one level inside the current element.
- ▶ **Expand Child Folds**
Unfolds all child elements of the currently selected element.
- ▶ **Expand All (Ctrl NumPad* (Command NumPad* on OS X))**
Unfolds all elements in the current document.

Using the Linking Support

When working on a suite of documents that reference one another (references, external entities, XInclude, DITA conref, etc), the **linking support** is useful for navigating between the documents. In the predefined customizations that are bundled with Oxygen XML Editor links are marked with an icon representing a chain link:  When hovering with the mouse pointer over the marker, the mouse pointer changes its shape to indicate that the link can be followed and a tooltip presents the destination location. Click the link to open the referenced resource in an editor. The same effect can be obtained by using the action **Document > File > Open file at caret (Ctrl Enter (Command Enter on OS X))** when the caret is in a link element.

 **Note:** Depending on the referenced file type, the **Open file at caret** action opens the target link either in the Oxygen XML Editor or in the default system application. If the target file does not exist, Oxygen XML Editor prompts you to create it.

Displaying the Markup

In the **Author** mode, you can control the amount of displayed markup using the following dedicated actions from the toolbar:

▶ Full Tags with Attributes

Displays full name tags with attributes for both block level as well as in-line level elements.

▶ Full Tags

Displays full name tags without attributes for both block level as well as in-line level elements.

▶ Block Tags

Displays full name tags for block level elements and simple tags without names for in-line level elements.

▶ Inline Tags

Displays full name tags for inline level elements, while block level elements are not displayed.

▶▷ Partial Tags

Displays simple tags without names for in-line level elements, while block level elements are not displayed.

✗ No Tags

No tags are displayed. This is the most compact mode.

To set a default mode of the tags mode, go to [Author preferences page](#) and configure the **Tags display mode** mode. However, if the document opened in Author editor does not have an associated CSS stylesheet, then the **Full Tags** mode will be used.

Block-level elements are those elements of the source document that are formatted visually as blocks (e. g. paragraphs), while the inline level elements are distributed in lines (e. g. emphasizing pieces of text within a paragraph, inline images, etc). The graphical format of the elements is controlled from the CSS sources via the `display` property.

Bookmarks

A position in a document can be marked with a bookmark. You can then quickly go to the marked position with a keyboard shortcut or a menu action. This is useful when navigating large documents or working on multiple documents where the cursor needs to move between several marked positions. The bookmarks are displayed with a small icon on the vertical strip to the left of the editor.

```

<xsl:output omit-xml-declaration="yes" method="html"/>
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="anchor">
  <a name="{@target}">
</xsl:template>
<xsl:template match="articledescription">
  <table width="100%" border="0" cellpadding="0" cellspacing="0">

```

Figure 34: Editor Bookmarks

A bookmark can be inserted by one of the following:

- Right-click on the vertical strip to the left of the editor.
- Select the **Create Bookmark (F9)** action from the **Edit > Bookmarks** menu.

A bookmark can be removed by right-clicking its icon on the vertical strip and select **Remove** or **Remove all (Ctrl+F7)** (you can also find the **Remove all (Ctrl+F7)** action in the **Edit > Bookmarks** menu).

You can navigate the bookmarks by using one of the actions available on the **Edit > Bookmarks > Go to** menu or by using the shortcut keys that are listed in that menu (for example, **Ctrl+1**, **Ctrl+2**). You can configure these shortcut keys from [Options > Menu Shortcut Keys](#).

Visual Hints for the Caret Position

When the caret is positioned inside a new context, a tooltip will be shown for a couple of seconds displaying the position of the caret relative to the current element context.

Here are the common situations that can be encountered:

- The caret is positioned before the first block child of the current node.

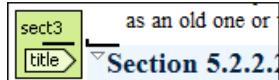


Figure 35: Before first block

- The caret is positioned between two block elements.

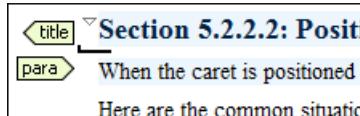


Figure 36: Between two block elements

- The caret is positioned after the last block element child of the current node.

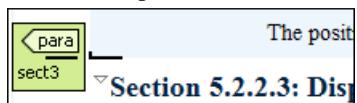


Figure 37: After last block

- The caret is positioned inside a node.

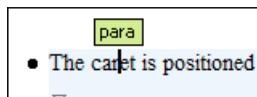


Figure 38: Inside a node

- The caret is positioned inside an element, before an inline child element.

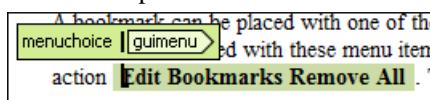


Figure 39: Before an inline element

- The caret is positioned between two inline elements.

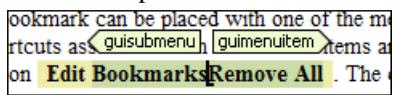


Figure 40: Between two inline elements

- The caret is positioned inside an element, after an inline child element.

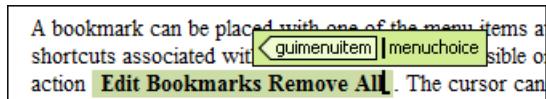


Figure 41: After an inline element

The nodes in the previous cases are displayed in the tooltip window using their names.

To deactivate this feature, [open the Preferences dialog box](#) and go to **Editor / Author > Show caret position tooltip**. Even if this option is **disabled**, you can trigger the display of the position tooltip by pressing **Shift+F2**.



Note: The position information tooltip is not displayed if one of the modes *Full Tags with Attributes* or *Full Tags* is selected.

Location Tooltip

When editing XML documents in a visual environment you might find it difficult to position the caret between certain tags that do not have a visual representation. To counterbalance this, Oxygen XML Editor displays a transparent preview of the Position Information Tooltip, called *Location Tooltip*:

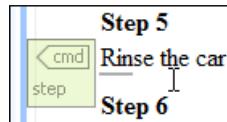


Figure 42: Location Tooltip

Oxygen XML Editor displays a location tooltip when the following conditions are met:

- you are editing the document in one of the following *tags display modes*: **Inline Tags**, **Partial Tags**, **No Tags**
- the mouse pointer is moved between block elements

To activate or deactivate this feature, use the **Show location tooltip on mouse move** option from the [Caret Navigation preferences page](#).

Displaying Referenced Content

The references to entities, XInclude, and DITA conrefs are expanded by default in Author mode and the referenced content is displayed. You can control this behavior from the [Author preferences page](#). The referenced resources are loaded and displayed inside the element or entity that refers them, however the displayed content cannot be modified directly.

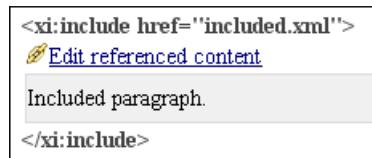


Figure 43: XInclude reference

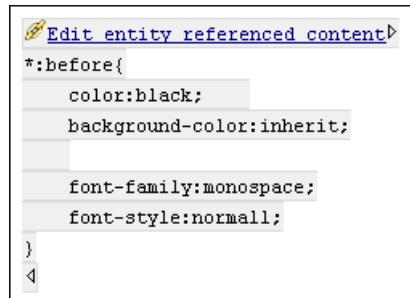


Figure 44: External entity reference

When the referenced resource cannot be resolved, an error will be presented inside the element that refers them instead of the content.

If you want to make modifications to the referenced content, you must open the referenced resource in an editor. The referenced resource can be opened quickly by clicking the link (marked with the icon ) which is displayed before the referenced content or by using the **Edit Reference** action from the contextual menu (in this case the caret is placed at

the precise location where the action was invoked in the main file). The referenced resource is resolved through the XML Catalog set in **Preferences**.

The referenced content is refreshed:

- Automatically, when it is modified and saved from Oxygen XML Editor.
- On demand, by using the [Refresh references action](#). Useful when the referenced content is modified outside the Oxygen XML Editor scope.

Finding and Replacing Text

You can search for a specific word or string of characters using the following features:

- [Find/Replace dialog box](#)
- [Find/Replace in Files dialog box](#)
- [Quick Find toolbar](#)
- [Find all Elements dialog box](#)

Complex search operations may take some time to complete. If a search operation takes more than 5 seconds, you are prompted to decide whether you want to continue the operation or stop it.

Contextual Menu

More powerful support for editing the XML markup is offered via actions included in the contextual menu. Two types of actions are available: **generic actions** (actions that not depends on a specific document type) and **document type actions** (actions that are configured for a specific document type).

The generic actions are:

Quick Fix (Alt 1 (Command Alt 1 on OS X))

Available when the contextual menu is invoked on an error where [Oxygen XML Editor can provide a quick fix](#).

Open Image

Available when the contextual menu is invoked on an image. This action allows you to open an image in the Oxygen XML Editor's [Image Viewer](#) or in a default system application associated with the current image type.

>Edit Attributes

A pop-up window is displayed allowing you to [manage in-place the element's attributes](#).

Edit Profiling Attributes...

Allows you to change the [profiling attributes](#) defined on all selected elements.

Cut, Copy, Paste

Common edit actions with the same functionality as those found in the text editor.

Paste special > Paste As XML

Similar to **Paste** operation, except that the clipboard's content is considered to be XML.

Paste special > Paste As Text

Pastes the clipboard content, ignoring any structure or styling markup, if any.

Select

Contains the following actions:

Select > Select Element

Selects the entire element at the current caret position.

Select > Select Content

Selects the entire content of the element at the current caret position, excluding the start and end tag. Performing this action repeatedly will result in the selection of the content of the ancestor of the currently selected element content.

Select > Select Parent

Selects the parent of the element at the current caret position.



Note: You can select an element by triple clicking inside its content. If the element is empty you can select it by double clicking it.

Text

Contains the following actions:

Text > To Lower Case

Converts the selected content to lower case characters.

Text > To Upper Case

Converts the selected content to upper case characters.

Text > Capitalize Sentences

Converts to upper case the first character of every selected sentence.

Text > Capitalize Words

Converts to upper case the first character of every selected word.

Text > Count Words

Counts the number of words and characters (no spaces) in the entire document or in the selection for regular content and read-only content.



Note: The content marked as deleted with *track changes* is ignored when counting words.

Refactoring

Contains a series of actions designed to alter the document's XML structure:

Toggle Comment

Encloses the currently selected text in an XML comment, or removes the comment if it is commented.

Move Up

Moves the current node or selected nodes in front of the previous node.

Move Down

Moves the current node or selected nodes after the successive node.

Split Element

Splits the content of the closest element that contains the caret's position. Thus, if the caret is positioned at the beginning or at the end of the element, the newly created sibling will be empty.

Join Elements

Joins two adjacent elements that have the same name. The action is available only when the caret position is between the two adjacent elements. Also, joining two elements can be done by pressing the **Delete** or **Backspace** keys and the caret is positioned between the boundaries of these two elements.

Surround with Tag...

Selected text in the editor is marked with the specified start and end tags.

Surround with <Tag name>

Selected text in the editor is marked with start and end tags used by the last **Surround with Tag...** action.

Rename Element

The element from the caret position and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog box.

Delete Element Tags

Deletes the tags of the closest element that contains the caret's position. This operation is also executed if the start or end tags of an element are deleted by pressing the **Delete** or **Backspace** keys.

Review

Provides access to *Track Changes* and *Manage Comments* actions.

Generate IDs

Provides access to *searching and refactoring actions for ID/IDREFS*.

Insert > Insert Entity

Allows the user to insert a predefined entity or a character entity. Surrogate character entities (range #x10000 to #x10FFFF) are also accepted. Character entities can be entered in one of the following forms:

- #<decimal value> - e. g. #65
- &#<decimal value>; - e. g. A
- #x<hexadecimal value> - e. g. #x41
- &#x<hexadecimal value>; - e. g. A

Options

Opens the [Author options page](#).

Document type actions are specific to some document type. Examples of such actions can be found in the [DocBook 4 Author Extensions](#) and [DITA Author Extensions](#) sections.

Editing XML Documents in Author Mode

This section details how to edit the text content and the markup of XML documents in **Author** mode. It also explains how to edit tables, images, MathML notations, and more, in **Author** mode.

Editing the XML Markup

One of the most useful feature in Author editor is the content completion support. The fastest way to invoke it is to press **Enter** or **Ctrl Space (Command Space on OS X)** in the editor panel.

Content completion window offers the following types of actions:

- Inserting allowed elements for the current context according to the associated schema.
- Inserting element values if such values are specified in the schema for the current context.
- Inserting new undeclared elements by entering their name in the text field.
- Inserting CDATA sections, comments, processing instructions.
- Inserting [code templates](#).
- If the **Show all possible elements in the content completion list** option from the [Schema aware preferences page](#) is enabled, the content completion pop-up window will present all the elements defined by the schema. When choosing an element from this section, the insertion will be performed using the schema aware smart editing features.

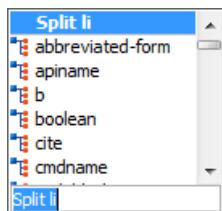


Figure 45: Content completion window

If you press **(Enter)** the displayed content completion window will contain as first entries the **Split <Element name>** items. Usually you can only split the closest block element to the caret position but if it is inside a list item, the list item will also be proposed for split. Selecting **Split <Element name>** splits the content of the specified element around the caret position. Thus, if the caret is positioned at the beginning or at the end of the element, the newly created sibling will be empty.

If the caret is positioned inside a space preserve element the first choice in the content completion window is **Enter** which inserts a new line in the content of the element. If there is a selection in the editor and you invoke content completion, a **Surround with** operation can be performed. The tag used will be the selected item from the content completion window.

By default you are not allowed to insert element names which are not defined by the schema. This can be changed by unchecking the **Allow only insertion of valid elements and attributes** check box from the [Schema aware preferences page](#).



Note: The content completion list of proposals contains elements depending on the elements inserted both before and after the caret position.

Joining two elements - You can choose to join the content of two sibling elements with the same name by using the **contextual menu > Join elements** action.

The same action can be triggered also in the next situations:

- The caret is located before the end position of the first element and **(Delete)** key is pressed.
- The caret is located after the end position of the first element and **(Backspace)** key is pressed.
- The caret is located before the start position of the second element and **(Delete)** key is pressed.
- The caret is located after the start position of the second element and **(Backspace)** key is pressed.

In either of the described cases, if the element has no sibling or the sibling element has a different name, **Unwrap** operation will be performed automatically.

Unwrapping the content of an element - You can unwrap the content of an element by deleting its tags using the **Delete element tags** action from the editor contextual menu.

The same action can be triggered in the next situations:

- The caret is located before the start position of the element and **(Delete)** key is pressed.
- The caret is located after the start position of the element and **(Backspace)** key is pressed.
- The caret is located before the end position of the element and **(Delete)** key is pressed.
- The caret is located after the end position of the element and **(Backspace)** key is pressed.

Removing all the markup of an element - You can remove the markup of the current element and keep only the text content by highlighting the appropriate block of content and use the **Remove All Markup** action that is available in the **Refactoring** submenu of the contextual menu and in the **Document > Markup** menu.

When you press **(Delete)** or **(Backspace)** in the presented cases the element is unwrapped or it is joined with its sibling. If the current element is empty, the element tags will be deleted.

When you click on a marker representing the start or end tag of an element, the entire element will be selected. The contextual menu displayed when you right-click on the marker representing the start or end tag of an element contains **Append child**, **Insert Before** and **Insert After** submenus as first entries.

Code Templates

Code templates are code fragments that can be inserted quickly at the current editing position . Oxygen XML Editor comes with a set of built-in code templates for CSS, LESS, Schematron, XSL, XQuery, and XML Schema document types. You can also [define your own code templates and share them with others](#).

To get a complete list of available code templates, press **Ctrl Shift Space (Command Shift Space on OS X)** in **Text** mode or **Enter** in **Author** mode. To enter the code template, select it from the list or type its code and press **Enter**. If a shortcut key has been assigned to the code template, you can also use the shortcut key to enter it. Code templates are displayed with a symbol in the content completion list.

When the **Content Completion Assistant** is invoked (**Ctrl Space (Command Space on OS X)**), it also presents a list of code templates specific to the type of the active editor.

To watch our video demonstration about code templates, go to http://oxygentools.com/demo/Code_Templates.html.

Editing the XML Content

By default you can type only in elements which accept text content. So if the element is declared as empty or element only in the associated schema you are not allowed to insert text in it. This is also available if you try to insert CDATA inside an element. Instead a warning message is shown:



Figure 46: Editing in empty element warning

You can disable this behavior by checking the **Allow Text in empty or element only content** check box in the [Author preferences page](#).

Entire sections or chunks of data can be moved or copied by using the drag and drop support. The following situations can be encountered:

- when both the drag and drop sources are Author pages, an well-formed XML fragment is transferred. The section is balanced before dropping it by adding matching tags when needed.
- when the drag source is the Author page but the drop target is a text-based editor only the text inside the selection is transferred as it is.
- the text dropped from another text editor or another application into the Author page is inserted without changes.

Removing the Text Content of the Current Element

You can remove the text content of the current element and keep only the markup by highlighting the appropriate block of content and use the **Remove Text** action that is available in the **Refactoring** submenu of the contextual menu and in the **Document > Markup** menu. This is useful when the markup of an element must be preserved, for example a table structure but the text content must be replaced.

Duplicating Elements with Existing IDs

If the **Auto generate IDs for elements** option (available in the **ID Options** dialog box from DITA, DocBook and TEI document types) is disabled and you duplicate elements with existing IDs, the duplicates lose these IDs. If the previously mentioned option is active, when you duplicate content, Oxygen XML Editor makes sure that if there is an ID attribute set in the XML markup, the newly created duplicate has a new, unique ID attribute value. The option **Remove IDs when copying content in the same document** allows you to control if a pasted element should retain its ID.

Table Layout and Operations

Oxygen XML Editor provides support for editing data in a tabular form. The following operations are available:

- **Adjusting column width**

You are able to manage table width and column width specifications from the source document. These specifications are supported both in fixed and proportional dimensions. The predefined frameworks (DITA, DocBook, and XHTML) also support this feature. The layout of the tables for these document types takes into account the table width and the column width specifications particular to them. To adjust the width of a column or table, drag the border of the column. The changes you make to a table are committed into the source document.

col[span:1, width:2*]	
col[span:1, width:0.5*]	
Person Name	Age
Jane	26
Bart	24
Alexander	22
They are all students of the computer science department	

Figure 47: Resizing a Column in Oxygen XML Editor Author Editor

- **Column and row selection**

To select a row or a column of a table, place the mouse cursor above the column or in front of the row you want to select, then click. When hovering the mouse cursor in front of rows or above column headers, the cursor changes to for row selection and to for column selection and that specific row or column is highlighted.

- **Cell selection**

To select a cell in a table, press and hold the **Ctrl** key and click anywhere inside the cell. You can use this action to select one or more cells, and also to deselect cells from a selection. Alternatively, you can click one of the left corners of a cell (right corners in case you are editing a *RTL document*). The cursor changes to when it hovers over the corners of the cell.

- **Rectangular selection**

To select a rectangular block of cells do one of the following:

- click a cell and drag to expand the selection
- click a cell, then press the **Shift** key and use the arrow keys to expand the selection

- **Drag and drop**

You can use the drag and drop action to edit the content of a table. You are able to select a column and drag it to another location in the table you are editing. When you drag a column and hover the cursor over a valid drop position, Oxygen XML Editor decorates the target location with bold rectangles. The same drag and drop action is also available for rows.

- **Copy-paste and cut for columns and rows**

In Oxygen XML Editor, you are able to copy entire rows or columns of the table you are editing. You can paste a copied column or row both inside the source table and inside other tables. The cut operation is also available for rows and columns. You can use the cut and the copy-paste actions for tables located in different documents as well.

When you paste a column in a non-table content, Oxygen XML Editor introduces a new table which contains the fragments of the source column. The fragments are introduced starting with the header of the column. When you copy a column of a CALS table, Oxygen XML Editor preserves the width information of the column. This information is then used when you paste the column in another CALS table.

- **Content deletion**

To delete a group of cells (can be columns, rows, or rectangular block of cells), select them and do one of the following:

- press either **Delete**, or **Backspace** on your keyboard to delete the cells' content. Press again **Delete**, or **Backspace** to remove the selected table structure
- if the selection is a column or a row, you can use the **Delete a table row** or **Delete a table column** actions to delete both the content and table structure

DocBook Table Layout

The DocBook table layout supports two models: CALS and HTML.

In the CALS table model, you can specify column widths using the `colwidth` attribute of the associated `colspec` element. The values can be fixed or proportional. By default, when you insert, drag and drop, or copy/paste a column, the value of the `colwidth` attribute is `1*`.

Also the `colsep` and `rowsep` attributes are supported. These control the way separators are painted between the table cells.

Sample CALS Table with no specified width and proportional column widths

colspecs...

column name	number	width	align	colsep	rowsep
c1	1	0.32*	▼	<input type="checkbox"/>	<input type="checkbox"/>
c2	2	1.49*	▼	<input type="checkbox"/>	<input type="checkbox"/>
c3	3	1.15*	▼	<input type="checkbox"/>	<input type="checkbox"/>
c4	4	0.4*	▼	<input type="checkbox"/>	<input type="checkbox"/>
c5	5	1.67*	▼	<input type="checkbox"/>	<input type="checkbox"/>

Horizontal Span		a3	a4	a5	
f1	f2	f3	f4	f5	
b1	b2	b3	b4	► Vertical [▲] Span	
c1					c4
d1	Spans ► Both [▲] directions				d5

Figure 48: CALS table in DocBook

Editing Table Component Properties

To customize the look of a table, place the caret anywhere in a table and invoke the **Table Properties (Ctrl T)** (**Command T on OS X**) action from one of the following locations:

- **contextual menu > Table** menu.
- **main menu > DocBook > Table**.
- **Table Properties** toolbar action.

The **Table properties** dialog box allows you to set specific properties to the table elements.



Note: Depending on the context, some options or values are filtered out.



Note: If you want to remove a property, set its value to **<not set>**.



Note: Choose the **<preserve>** setting to:

- keep the current non-standard value for a particular property.
- keep the values already set. This happens when you select multiple elements having the same property set to different values.

For a **CALS** table you can format any of the following:

- **Table** - set the horizontal alignment, row and column separators and the table's frame.
- **Row** - set the row type, vertical alignment and row separator.
- **Column** - set the horizontal alignment, and column and row separators.
- **Cell** - set the horizontal and vertical alignment, column and row separators.

For an **HTML** table you can customize any of the following:

- **Table** - set the frame attribute.
- **Row** - set the row type, horizontal and vertical alignment.
- **Column** - set the horizontal and vertical alignment.
- **Cell** - set the horizontal and vertical alignment.

XHTML Table Layout

The HTML table model accepts both table and column widths. Oxygen XML Editor uses the `width` attribute of the `table` element and the `col` element associated with each column. Oxygen XML Editor displays the values in fixed units, proportional units, or percentages.

colspe...

A table with merged cells, fixed column widths, and fixed total width.

x	y	Spans ▴ Horizontally			
Spans ▴ Vertically		b			
↳ Spans ▴ Both		d		e f	
g	h	i	k		

Figure 49: HTML table

DITA Table Layout

Depending on the context, the DITA table layout accepts CALS tables, simple tables, and choice tables.

In the CALS table model, you can specify column widths using the `colwidth` attribute of the associated `colspec` element. The values can be fixed or proportional. By default, when you insert, drag and drop, or copy/paste a column, the value of the `colwidth` attribute is `1*`.

Also the `colsep` and `rowsep` attributes are supported. These control the way separators are painted between the table cells.

▼ *Sample CALS Table with no specified width and proportional column widths*

▼ colspe...

column name c1	number 1	width 0.32*	▼	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name c2	number 2	width 1.49*	▼	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name c3	number 3	width 1.15*	▼	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name c4	number 4	width 0.4*	▼	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name c5	number 5	width 1.67*	▼	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
Horizontal Span		a3	a4	a5	
f1	f2	f3	f4	f5	
b1	b2	b3	b4	↳ Vertical ▲ Span	
c1				c4	
d1	Spans ▴ Both ▲ directions			d4	d5

Figure 50: CALS table in DITA

The simple tables accept only relative column width specifications by using the `relcolwidth` attribute of the `simpletable` element.

Header 1	Header 2
Column 1	Column 2

Figure 51: DITA simple table

You can insert choice tables in DITA tasks either using the **Content Completion Assistant** or using the toolbar and contextual menu actions.

Editing Table Component Properties

To customize the look of a table, place the caret anywhere in a table and invoke the  **Table Properties (Ctrl T)** (**Command T on OS X**) action from one of the following locations:

- contextual menu > **Table** menu
- main menu > **DITA** > **Table**
-  **Table Properties** toolbar action

The **Table properties** dialog box allows you to set specific properties to the table elements.

 **Note:** Depending on the context, some options or values are filtered out.

 **Note:** If you want to remove a property, set its value to **<not set>**.

 **Note:** Choose the **<preserve>** setting to:

- keep the current non-standard value for a particular property.
- keep the values already set. This happens when you select multiple elements having the same property set to different values.

For a **CALS** table you can format any of the following:

- **Table** - set the horizontal alignment, row and column separators and the table's frame.
- **Row** - set the row type, vertical alignment and row separator.
- **Column** - set the horizontal alignment, and column and row separators.
- **Cell** - set the horizontal and vertical alignment, column and row separators.

For a *simple* table you can customize any of the following:

- **Table** - set the frame attribute.
- **Row** - set the row type.

Sorting Content in Tables and List Items

Oxygen XML Editor offers support for sorting the content of tables and list items of ordered and unordered lists.

What do you want to do?

- *Sort an entire table.*
- *Sort a selection of rows in a table.*
- *Sort a table that contains cells merged over multiple rows.*
- *Sort a table based on multiple sorting criteria.*
- *Sort list items.*

Sorting a Table

To sort rows in a table, select the entire table (or specific rows) and use the  **Sort** action from the main toolbar or the contextual menu. This opens the **Sort** dialog box.

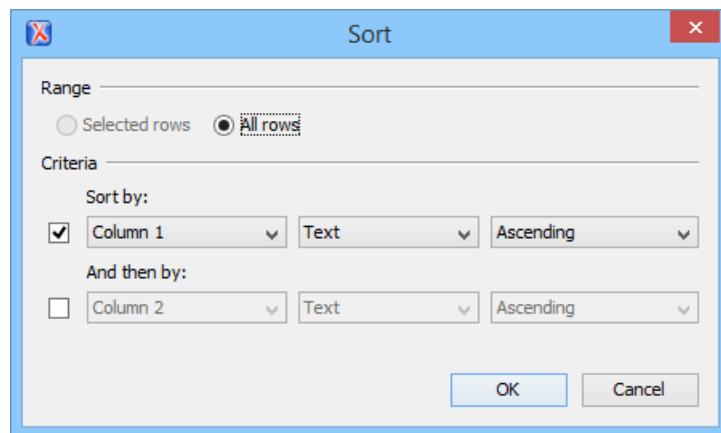


Figure 52: The "Sort" Dialog Box

This dialog box sets the range that is sorted and the sorting criteria. The range is automatically selected depending on whether you sort an entire table or only a selection of its rows.

 **Note:** When you invoke the sorting operation over an entire table, the **Selected rows** option is disabled.

The **Criteria** section specifies the sorting criteria (a maximum of three sorting criteria are available), defined by the following:

- A name, which is collected from the column heading.
- The type of the information that is sorted (either text, numeric, or date).
- The sorting direction (either ascending or descending).

The sort criteria is automatically set to the column where the caret is located at the time when the sorting operation is invoked.

 **Note:** The sorting mechanism of Oxygen XML Editor recognizes multiple date formats, such as *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*.

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. If you want to revert to the initial order of your content, press **Ctrl Z (Command Z on OS X)** on your keyboard.

 **Note:** The sorting support takes the value of the `xml:lang` attribute into account and sorts the content in a natural order.

Sorting a Selection of Rows

To sort a selection of rows in a table, select the rows that you want to sort and either right click the selection and choose  **Sort**, or click  **Sort** on the main toolbar. This opens the **Sort** dialog box.

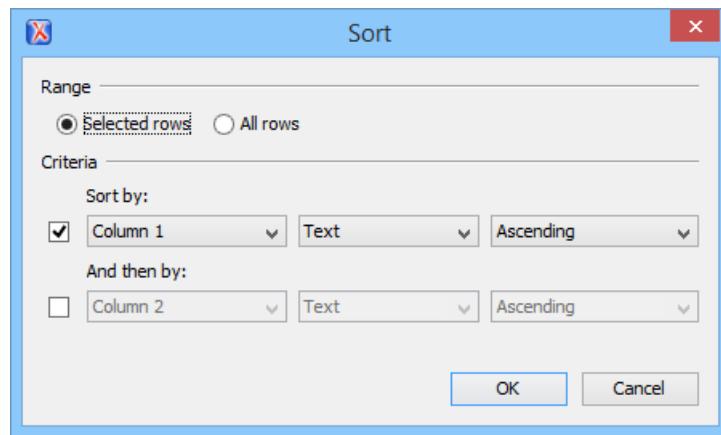


Figure 53: Sort Selected Rows

This dialog box sets the range that is sorted and the sorting criteria. The range is automatically selected depending on whether you sort an entire table or only a selection of its rows.

The **Criteria** section specifies the sorting criteria (a maximum of three sorting criteria are available), defined by the following:

- A name, which is collected from the column heading.
- The type of the information that is sorted (either text, numeric, or date).
- The sorting direction (either ascending or descending).

The sort criteria is automatically set to the column where the caret is located at the time when the sorting operation is invoked.

 **Note:** The sorting mechanism of Oxygen XML Editor recognizes multiple date formats, such as *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*.

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. If you want to revert to the initial order of your content, press **Ctrl Z (Command Z on OS X)** on your keyboard.



Note: The sorting support takes the value of the `xml:lang` attribute into account and sorts the content in a natural order.

Sorting a Table that Contains Merged Cells

In case a table contains cells that span over multiple rows, you can not perform the sorting operation over the entire table. Still, the sorting mechanism works over a selection of rows that do not contain `rowspans`.



Note: For this type of table, the **Sort** dialog keeps the **All rows** option disabled even if you perform the sorting operation over a selection of rows.

Sorting Using Multiple Criteria

You can sort both an entire table or a selection of its rows based on multiple sorting criteria. To do so, enable the rest of the criteria in the **Sort** dialog, configure the applicable items and click **OK** to complete the sorting operation.

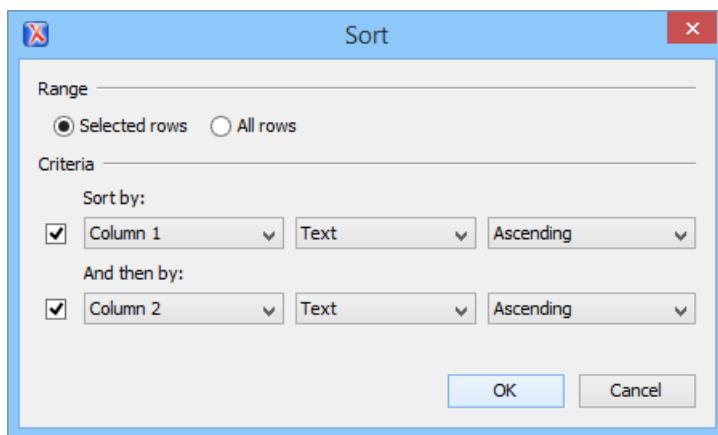


Figure 54: Sorting Based on Multiple Criteria

Sorting List Items

A sorting operation can be performed on various types of lists and list items. Oxygen XML Editor provides support for sorting the following types of lists:

- Ordered list (`ol`)
- Unordered list (`ul`)
- Parameter list (`paraml`)
- Simple list (`s1`)
- Required conditions (`reqconds`)
- Supplies list (`supplyli`)
- Spare parts list (`sparesli`)
- Safety conditions (`safety`)
- Definitions list (`d1`)

The sorting mechanism works on an entire list or on a selection of list items. To sort items in a list, select the items or list and use the **Sort** action from the main toolbar or the contextual menu. This opens the **Sort** dialog box.

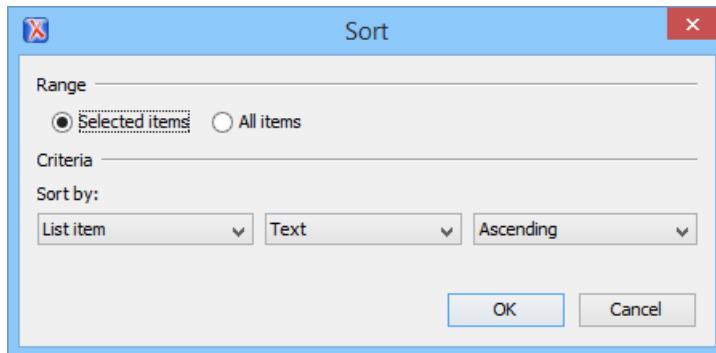


Figure 55: Sorting List Items

This dialog box sets the range that is sorted and the sorting criteria. The range is automatically selected depending on whether you sort an entire list or only a selection of its items.

 **Note:** When you invoke the sorting operation over an entire list, the **Selected rows** option is disabled.

The **Criteria** section specifies the sorting criteria, defined by the following:

- The name of the type of item being sorted.
- The type of information that is sorted (*text*, *numeric*, or *date*).
- The sorting direction (*ascending* or *descending*).

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. If you want to revert to the initial order of your content, press **Ctrl Z (Command Z on OS X)** on your keyboard.

 **Note:** The sorting support takes the value of the `xml:lang` attribute into account and sorts the content in a natural order.

Image Rendering

The **Author** mode and the output transformation process might render the images referenced in an XML document differently, since they use different rendering engines.

Table 4: Supported Image Formats

Image Type	Support	Additional Information
GIF	built-in	Animations not yet supported
JPG, JPEG	built-in	<i>JPEG images with CMYK color profiles</i> are properly rendered only if color profile is inside the image.
PNG	built-in	
SVG, SVGZ, WMF	built-in	Rendered using the open-source Apache Batik library which supports SVG 1.1.
BMP	built-in	
TIFF	built-in	Rendered using a part of the Java JAI Image library.
EPS	built-in	Renders the preview TIFF image inside the EPS.
AI	built-in	Renders the preview image inside the Adobe Illustrator file.
JPEG 2000, WBMP	plug-in	Renders by <i>installing the Java Advanced Imaging (JAI) Image I/O Tools plug-in</i> .
CGM	plug-in	Renders by <i>installing an additional library</i> .
PDF	plug-in	Renders by <i>installing the Apache PDF Box library</i> .

When an image cannot be rendered, Oxygen XML Editor **Author** mode displays a warning message that contains the reason why this is happening. Possible causes include the following:

- The image is too large. Enable the *Show very large images* option.
- The image format is not supported by default. It is recommended to *install the Java Advanced Imaging(JAI) Image I/O Tools plug-in*.

Scaling Images

Image dimension and scaling attributes are taken into account when an image is rendered. The following rules apply:

- If you specify only the width attribute of an image, the height of the image is proportionally applied.
- If you specify only the height attribute of an image, the width of the image is proportionally applied.
- If you specify width and height attributes of an image, both of them control the rendered image.
- If you want to scale both the width and height of an image proportionally, use the *scale* attribute.

 **Note:** As a Java application, Oxygen XML Editor uses the *Java Advanced Imaging API* that provides a pluggable support for new image types. If you have an *ImageIO* library that supports additional image formats, just copy this library to the `[OXYGEN_DIR]/lib` directory.

Installing Java Advanced Imaging(JAI) Image I/O Tools Plug-in

Follow this procedure:

1. Start Oxygen XML Editor and open the **Help > About** dialog box. Go to the **System properties** tab and look for the *java.runtime.name* and *java.home* properties. Keep their values for later use.
 2. *Download* the JAI Image I/O kit corresponding to your operating system and Java distribution (found in the *java.runtime.name* property).
- Please note that the JAI API is not the same thing as JAI Image I/O. Make sure you have installed the latter.
3. Execute the installer. When the installation wizard displays the **Choose Destination Location** page, fill-in the **Destination Folder** field with the value of the *java.home* property. Continue with the installation procedure and follow the on-screen instructions.

OS X Workaround

There is no native implementation of JAI Image I/O for OS X 10.5 and later. However, the JAI Image I/O has a Java implementation fallback which also works on OS X. Some of the image formats are not fully supported in this fallback mode, but at least the TIFF image format is known to be supported.

1. Download a Linux(tar.gz) distribution of JAI Image I/O from:
<http://download.java.net/media/jai-imageio/builds/release/1.1/> e.g.
`jai_imageio-1_1-lib-linux-amd64.tar.gz`
2. In the `[OXYGEN_DIR]/lib` directory create a directory named `endorsed` e.g.
`[OXYGEN_DIR]/lib/endorsed`.
3. Unpack the tar.gz and navigate to the lib directory from the unpacked directory. e.g. `jai_imageio-1_1/lib`. Copy the jar files from there(`clibwrapper_jiio.jar` and `jai_imageio.jar`) to the `[OXYGEN_DIR]/lib/endorsed` directory.
4. Restart the application and the JAI Image I/O support will be up and running.

Customize Oxygen XML Editor to Render CGM Images (Experimental Support)

Oxygen XML Editor provides experimental support for CGM 1.0 images.



Attention: Image hotspots are not supported.

Since this is an experimental support, some graphical elements might be missing from the rendered image.

The CGM rendering support is based on a third party library. In its free of charge variant it renders the images watermarked with the string Demo, painted across the panel. You can find more information about ordering the fully functioning version here: <http://www.bdaum.de/cgmpanel.htm>.

Follow this procedure to enable the rendering of CGM images in **Author** mode:

1. Download the `CGMPANEL.ZIP` from <http://www.bdaum.de/CGMPANEL.ZIP>.
2. Unpack the ZIP archive and copy the `cgmpanel.jar` into the `[OXYGEN_DIR]\lib` directory.
3. Restart the application.

Customize Oxygen XML Editor to Render PDF Images (Experimental Support)

Oxygen XML Editor provides experimental support for PDF images using the Apache PDFBox library.

Follow this procedure to enable the rendering of PDF images in **Author** mode:

1. Go to <http://pdfbox.apache.org/downloads.html> and download the pre-built PDFBox standalone binary JAR files `pdfbox-1.8.9.jar`, `fontbox-1.8.9.jar`, and `jempbox-1.8.9.jar`.
2. Copy the downloaded JAR libraries to the `[OXYGEN_DIR]\lib` directory.
3. Restart the application.

Customize Oxygen XML Editor to Render EPS and AI Images

Most EPS and AI image files include a preview picture of the content. Oxygen XML Editor tries to render this preview picture. The following scenarios are possible:

- The EPS or AI image does not include the preview picture. Oxygen XML Editor cannot render the image.
- The EPS image includes a TIFF preview picture.



Note: Some newer versions of the TIFF picture preview are rendered in gray-scale.

- The AI image contains a JPEG preview picture. Oxygen XML Editor renders the image correctly.

Adding an Image

To insert an image in a document while editing in **Author** mode, use one of the following methods:

- Click the **Insert Image Reference** action from the toolbar and choose the image file you want to insert. Oxygen XML Editor tries to reference the image with a path that is relative to that of the document you are currently editing. For example, if you want to add the `file:/C:/project/xml/dir/img1.jpg` image into `file:/C:/project/xml/doc1.xml` document, Oxygen XML Editor inserts a reference to `dir/img1.jpg`. This is useful when multiple users work on a common project and they have it stored in different locations in their computers.



Note: The **Insert Image Reference** action is available for the following document types: DocBook 4, DocBook 5, DITA, TEI P4, TEI P5, XHTML.

- Drag an image from other application and drop it in the **Author** editor. If it is an image file, it is inserted as a reference to the image file. For example, in a DITA topic the path of the image file is inserted as the value of the `href` attribute in an `image` element:

```
<image href="..../images/image_file.png"/>
```



Note: To replace an image, just drag and drop a new image over the existing one. Oxygen XML Editor will automatically update the reference to the new image.

- Copy the image from another application (such as an image editor) and paste it into your document. Oxygen XML Editor prompts you to first save it. After saving the image, a reference to that file path is inserted at the paste position.

Editing MathML Notations

The **Author** editor includes a built-in editor for *MathML* notations. To start the *MathML* editor, either double-click a *MathML* notation, or select the **Edit Equation** action from its contextual menu. In the *MathML* editor you are able to edit the mathematical symbols of a *MathML* notation. You can open a *MathML* file of your current project directly in the *MathML* editor. To do this, select **Open with > MathML editor** from the contextual menu in the **Project** view.

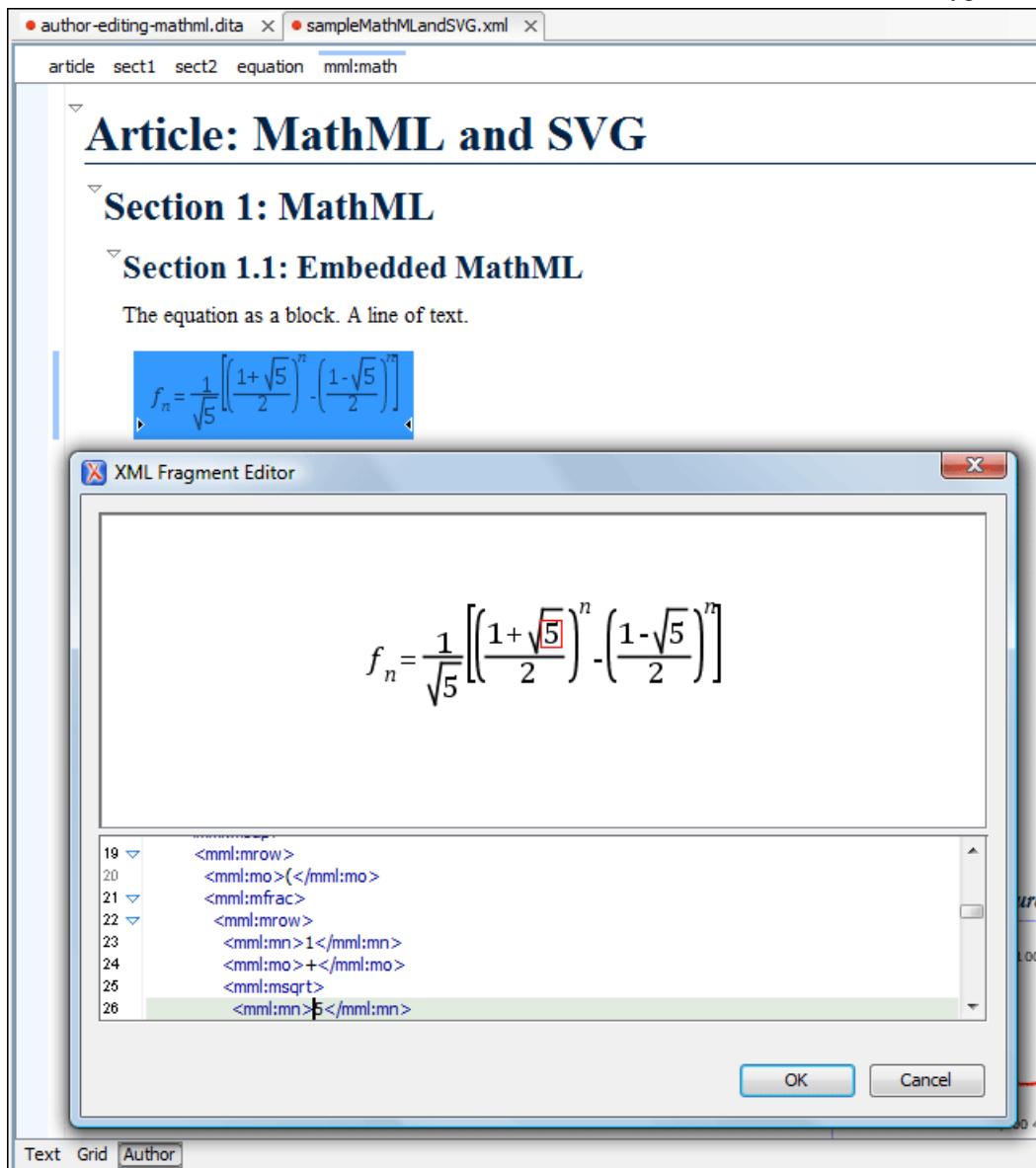


Figure 56: The default *MathML* editor

The font size and font family that is used for the equations is based upon the context in which the MathML equation appears. To configure the minimum font size of the equation, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > MathML**.

Configure the *MathFlow* Editor

The *MathFlow* Components product (*MathFlow SDK*) can replace the default *MathML* editor with a specialized *MathML* editor. You have to [purchase a MathFlow Component from Design Science](#) and configure it in Oxygen XML Editor with the following procedure:

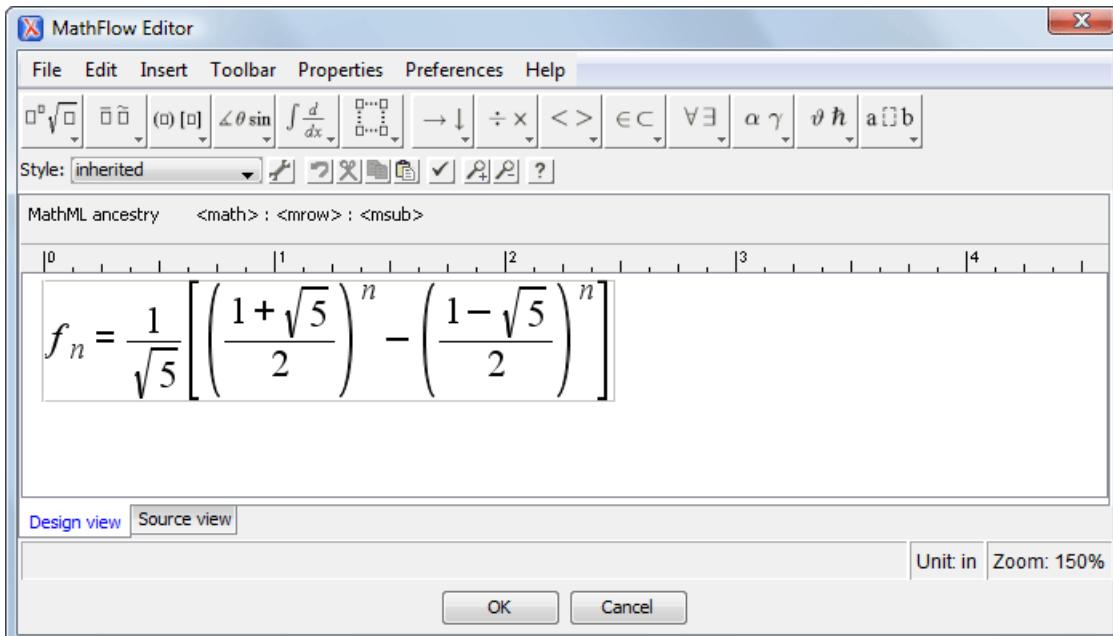


Figure 57: The default MathML editor

1. Install *MathFlow Components (MathFlow SDK)*.
2. On Windows make sure there is a copy of the *FLEXlm* DLL, which is the file [MathFlow-install-folder]/resources/windows/lmgr10.dll, in a folder that is added to the *PATH* environment variable.
3. Set the path to the *MathFlow* install folder *in the Preferences*.
4. Set the path to the *MathFlow* license file *in the Preferences*.

Refreshing the Content

On occasion you may need to reload the content of the document from the disk or reapply the CSS. This can be performed by using the **Reload** action.

To refresh the content of the referenced resources you can use the **Refresh references** action. However, this action will not refresh the expanded external entities, for which you will need to use the **Reload** action.

Presenting Validation Errors

Automatic validation and validate on request operations are available while editing documents in the **Author** mode. A detailed description of the document validation process and its configuration is described in the [Validating Documents section](#).

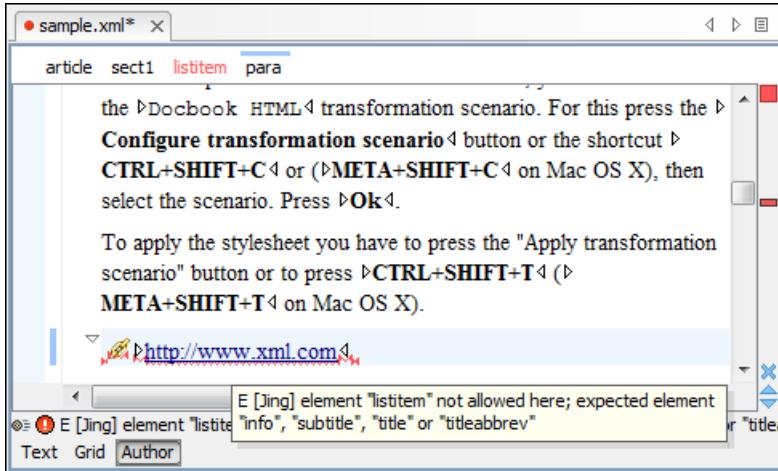


Figure 58: Presenting Validation Errors in Author Mode

A fragment with a validation error is marked by underlining the error in red, and validation warnings are underlined in yellow.

Also, the ruler on the right side of the editor panel is designed to display the errors found during the validation process and to help the user locate them in the document. The ruler contains the following:

- The top area - A success indicator square will turn green if the validation is successful, red if validation errors are found, or yellow if validation warnings are found. More details about the errors or warnings are displayed in a tool tip when you hover over indicator square. If there are numerous errors, only the first three are presented in the tool tip.
- The middle area - Errors are depicted with red markers, and warnings with yellow markers. If you want to limit the number of markers that are shown, [open the Preferences dialog box](#) and go to **Editor > Document checking > Maximum number of validation highlights**.

Clicking on a marker will highlight the corresponding text area in the editor. The error or warning message is also displayed both in a tool tip (when hovering over the marker) and in the message area on the bottom of the editor panel.

The validation status area at the bottom of the editor panel presents the message of the current validation error.

Clicking on the **Document checking options** button opens the [Document checking user preferences](#) page

- The bottom area - Two navigation arrows () allow you to skip to the next or previous error. The same actions can be triggered from **Document > Automatic validation > Next error (Ctrl . (Command , on OS X))** and **Document > Automatic validation > Previous error (Ctrl , (Command , on OS X))**. Also, the button can be used to clear all the error markers.

Status messages from every validation action are logged in the [Information view](#).

Author Whitespace Handling

When you edit a document in Author mode, Oxygen XML Editor must serialize the resulting document as XML. Oxygen XML Editor serializes the document when you save it or switch to another editing mode. When the document is serialized, Oxygen XML Editor [formats and indents the XML document](#) according to the current [format and indent settings](#).

Minimizing whitespace differences between versions

When serializing a document to XML, Author mode will only format and indent those elements of the document that have been edited. Any element that has not been edited will be serialized exactly as it was loaded from disk. This is useful when your content is managed in a version control systems, as it avoids introducing insignificant whitespace differences between version, which in turn makes diff output easier to read.

Entering whitespace in Author mode

Oxygen XML Editor controls the entry of whitespace characters in Author mode according the [XML whitespace rules](#), which means it will not let you insert insignificant whitespace. This means that it will not let you insert extra line-breaks or spaces inside a typical paragraph element, for instance. (Any such whitespace would be normalized away when the document was serialized to XML, so Oxygen XML Editor is saving you from any surprises when this happens.)

Of course, you will legitimately want to enter additional spaces and returns in some cases, such as code samples. Oxygen XML Editor will allow this in elements that are configured as preserve space elements according to the XML whitespace rules. For all of its [predefined document types](#), Oxygen XML Editor is [correctly configured to recognize preserve space elements](#) and to allow you to enter additional spaces in them.

If you are using a predefined document type and you are unable to enter additional whitespace, make sure that you are using an element from that document type that is intended to be a preserve-space element.

If you are using a custom document type, make sure that it is [configured correctly](#) so that Oxygen XML Editor recognizes that the current element is a preserve-space element.

Review

Tracking Document Changes

Track Changes is a way to keep track of the changes you make to a document. To activate track changes for the current document, either choose **Edit > Review > Track Changes** or click the  **Track Changes** button on the **Review** toolbar. When **Track Changes** is enabled, your modifications are highlighted using a distinctive color. The name of the author who is currently making changes and the colors can be customized from the [Review preferences page](#).

Docbook 4 supports ▷XHTML

Person Name	Age
Jane	26
Bart	24
Alexander	22
John	25

 tables:

Sample XHTML Table with fixed width and proportional column widths

```
col[span:1, width:2.08*]
col[span:1, width:0.46*]
```

Person Name	Age
Jane	26
Bart	24
Alexander	22
John	25

▷They belong are all students of the computer science department

Inserted by John Doe
Wed Apr 08 16:10:32 EEST 2009

This is a list of useful ▷XML links:

Figure 59: Change Tracking in Author Mode

When hovering over a change the tooltip displays information about the author and modification time.

Track Changes highlights textual changes and also changes that you make to the attributes in a document. Here is the list of tracked changes:

- Inserting, deleting content (text or elements)
- Drag and drop content (text or elements)
- Cutting or pasting content (text or elements)
- Inserting, deleting, and changing the structure of tables
- Inserting and editing lists and their content
- Inserting and deleting entities
- Inserting and deleting element tags
- Editing attributes
- Performing a **Split** operation
- Performing a **Surround with** operation

If the selection in the **Author** view contains tracked changes and you are copying it, the clipboard contains the selection with all the *accepted* changes. This filtering is performed only if the selection is not entirely inside a tracked change. The changes are stored in the document as processing instructions and they do not interfere with validating and transforming it. For each change, the author name and the modification time are preserved.

The following processing instruction is an example of how an *insert* change is stored in a document:

```
<?oxy_insert_start author="John Doe"
timestamp="20090408T164459+0300"?>all<?oxy_insert_end?>
```

The following processing instruction is an example of how an *delete* change is stored in a document:

```
<?oxy_delete author="John Doe" timestamp="20090508T164459+0300"
content="belong"?>
```



Note: Tracked changes are also shown in the **Outline** view. Deleted content is rendered with a strike through.

Adding Document Comments

You can associate a note or a comment to a selected area of content. Comments can highlight virtually any content from your document, except *read-only* text. The difference between such comments and change tracking is that a comment can be associated to an area of text without modifying or deleting the text.

The actions for managing comments are **Add Comment**, **Edit Comment**, **Delete Comment** and **Manage Comments** and are available on the **Review** toolbar and on the **Review** submenu of the contextual menu of the Author editor.



Tip: The comments are stored in the document as processing instructions, containing information about the author name and the comment time:

```
<?oxy_comment_start author="John Doe" timestamp="20090508T164459+0300" comment="Do not change this
content"?>
    Important content
<?oxy_comment_end?>
```

Comments are persistent highlights with a colored background. The background color is customizable or can be assigned automatically by the application. This behavior can be controlled from the [Review preferences page](#).



Note: Oxygen XML Editor presents the tracked changes in DITA conrefs and XInclude fragments.

Managing Changes

You can review the changes you or other authors made and then accept or reject them using the **Track Changes** toolbar



, or the similar actions from the **Edit > Review** menu:

Track Changes

Enables or disables the track changes support for the current document.

Accept Change(s)

Accepts the change located at the caret position. If you select a part of a delete or insert change, then only the selected content is accepted. If you select multiple changes, all of them are accepted. For an insert change, it means keeping the inserted text and for a delete change it means removing the content from the document.

Reject Change(s)

Rejects the change located at the caret position. If you select a part of a delete or insert change, then only the selected content is rejected. If you select multiple changes, all of them are rejected. For an insert change, it means removing the inserted text and for a delete change it means preserving the original content from the document.

Comment Change

You can decide to add additional comments to an already existing change. The additional description appears in the tooltip when hovering over the change and in the **Manage Tracked Changes** dialog box when navigating changes.

Highlight

Enables or disables the **Highlight tool**. Use the **Highlight** drop-down list to select a new color.

Add Comment

Inserts a comment in the document you are editing, at the caret position.

Edit Comment

Edits a selected comment from the edited document.

Remove Comment

Removes a selected comment from the edited document.

Manage Reviews

Opens the **Review view**.

-Track Changes Visualization Modes Drop-Down List

This drop-down list includes specialized actions that allow you to switch between the following visualization modes:

- **View All Changes/Comments** - This mode is active by default. When you use this mode, all tracked changes are represented in the Author mode.
- **View only Changes/Comments by** - Only the tracked changes made by the author you select are presented.
- **View Final** - This mode offers a preview of the document as if all tracked changes (both inserted and deleted) were accepted.
- **View Original** -this mode offers a preview of the document as if all tracked changes (both inserted and deleted) were rejected. You cannot edit the document in this mode. Attempting to do so switches the view mode to **View All Changes**.

All four actions are available only in the drop-down list in the **Review** toolbar. If you use **View Final** mode and **View Original** mode, highlighted comments are not displayed. To display highlighted comments, use **View All Changes/Comments**.

To watch our video demonstration about the Track Changes support, go to
http://oxygentools.com/demo/Change_Tracking.html.

Track Changes Behavior

This section explains the behaviour of the **Track Changes** feature depending on the context and whether it is activated.

You can use the **Track Changes** feature to keep track of multiple actions.

Possible change tracking scenarios:

- *Inserted content*
- *Surrounded content*
- *Deleted characters*
- *Deleted content*
- *Copied content*
- *Pasted content*
- *Attribute changes*

Keep Tracking of Inserted Content

When **Track Changes** is disabled and you insert content, the following cases are possible:

- Making an insertion in a **Delete** change - the change is split in two and the content is inserted without being marked as change.
- Making an insertion in an **Insert** change - the change is split in two and the content is inserted without being marked as change.
- Making an insertion in regular content - regular insertion.

When **Track Changes** is enabled and you insert content, the following cases are possible:

- Making an insertion in a **Delete** change - the change is split in two and the current inserted content appears marked as an **INSERT**.
- Making an insertion in an **Insert** change:
 - If the original insertion was made by another user, the change is split in two and the current inserted content appears marked as an **INSERT** by the current author.
 - If the original **Insert** change was made by the same user, the change is just expanded to contain the inserted content. The creation time-stamp of the previous insert is preserved.
- If we insert in regular content, the current inserted content appears marked as an **Insert** change.

Keep Tracking of Surrounded Content

When **Track Changes** is enabled and you surround content in a new XML element, the following cases are possible:

- Making a surround in a **Delete** change - nothing happens.
- Making a surround in an **Insert** change:
 - If the original insertion was made by another user, the change is split in two and the surround operation appears marked as being performed by the current author.
 - If the original **Insert** change was made by the same user, the existing change is just expanded to contain the surrounded content.
- Making a surround in regular content - the operation is marked as a surround change.

Keep Tracking of Deleted Characters

When **Track Changes** is disabled and you delete content character by character, the following cases are possible:

- Deleting content in an existing **Delete** change - nothing happens.
- Deleting content in an existing **Insert** change - the content is deleted without being marked as a deletion and the **INSERT** change shrinks accordingly.
- Deleting in regular content - regular deletion.

When **Track Changes** is enabled and you delete content character by character, the following cases are possible:

- Deleting content in an existing **Delete** change:

- If the same author created the **Delete** change, the previous change is marked as deleted by the current author.
- If another author created the **Delete** change, nothing happens.
- Deleting content in an existing **Insert** change:
 - If the same author created the **Insert** change, the content is deleted and the **Insert** change shrinks accordingly.
 - If another author created the **Insert** change, the **Insert** change is split in two and the deleted content appears marked as a **Delete** change by the current author.
- Deleting in regular content - the content is marked as **Delete** change by the current author.

Keep Tracking of Deleted Content

When **Track** changes is disabled and you delete selected content, the following cases are possible:

- The selection contains an entire **Delete** change - the change disappears and the content is deleted.
- The selection intersects with a **Delete** change (starts or ends in one) - nothing happens.
- The selection contains an entire **Insert** change - the change disappears and the content is deleted.
- The selection intersects with an **Insert** change (starts or ends in one), the **Insert** change is shrieked and the content is deleted.

When **Track** changes is enabled and you delete selected content, the following cases are possible:

- The selection contains an entire **Delete** change - the change is considered as rejected and then marked as deleted by the current author, along with the other selected content.
- The selection intersects a **Delete** change (starts or ends in one) - the change is considered as rejected and marked as deleted by the current author, along with the other selected content.
- The selection contains an entire **Insert** change:
 - If the **Insert** is made by the same author, the change disappears and the content is deleted.
 - If the **Insert** is made by another author, the change is considered as accepted and then marked as deleted by the current author, along with the other selected content.
- If the selection intersects an **Insert** change (starts or ends in one), the **Insert** change shrinks and the part of the **Insert** change that intersects with the selection is deleted.

Keep Tracking of Copied Content

When **Track Changes** is disabled and you copy content the following cases are possible:

- If the copied area contains **Insert** or **Delete** changes, these are also copied to the clipboard.

When **Track Changes** is enabled and you copy content the following cases are possible:

- If the copied area contains **Insert** or **Delete** changes, these are all accepted in the content of the clipboard (the changes will no longer be in the clipboard).

Keep Tracking of Pasted Content

When **Track Changes** is disabled and you paste content the following cases are possible:

- If the clipboard content contains INSERT OR DELETE changes, they will be preserved on paste.

When **Track Changes** is enabled and you paste content the following cases are possible:

- If the clipboard content contains **Insert** or **Delete** changes, all the changes are accepted and then the paste operation proceeds according to the insertion rules.

Keep Tracking of Attribute Changes

The **Track Changes** feature is able to keep the track of changes you make to attributes in a document. If the [Callouts support is enabled](#), all the attribute changes are presented as callouts in the document you are editing. The changes are also presented in the [Review view](#) and [Attributes view](#).

When you copy a fragment that contains tracked attribute changes, the following cases are possible:

- If you perform the copy operation with **Track Changes** enabled, all the attribute changes in the fragment are accepted.
- If you perform the copy operation with **Track Changes** disabled, the fragment holds the attribute changes inside it.

When you paste a fragment that contains tracked attribute changes, the following cases are possible:

- If you perform the paste operation with **Track Changes** enabled, the changes are accepted before the paste operation.
- If you perform the paste operation with **Track Changes** disabled, the changes are pasted in the document.

Track Changes Limitations

Recording changes has limitations and there is no guarantee that rejecting all changes will return the document to exactly the same state in which it originally was. Recorded changes are not hierarchical, a change cannot contain other changes inside. For example, if you delete an insertion made by another user, then reject the deletion, the information about the author who made the previous insertion is not preserved.

Track Changes Markup

Depending on the type of your edits, the following track changes markup appears in a document when you activate the



Track Changes feature:

Edit Type	Processing Instruction Start Marker	Processing Instruction End Marker	Attributes
Insertion	<?oxy_insert_start?>	<?oxy_insert_end?>	author, timestamp
Split	<?oxy_insert_start?>	<?oxy_insert_end?>	author, timestamp, type="split"
Surround	<?oxy_insert_start?>	<?oxy_insert_end?>	author, timestamp, type="surround"
Deletion	<?oxy_delete?>	-	author, timestamp, content
Comment	<?oxy_comment_start?>	<?oxy_comment_end?>	author, timestamp, comment, mid
Attribute Change	<?oxy_attributes?>	-	id, type, oldValue, author, timestamp

In case a comment intersects another, the mid attribute is used to correctly identify start and end processing instruction markers.

Intersecting Comments Markup

```
<?oxy_comment_start author="Andrew" timestamp="20130111T151520+0200" comment="Do we have a
task about pruning trees?"?>Unpruned
    <?oxy_comment_start author="Matthew" timestamp="20130111T151623+0200" comment="What time
of the year do they flower?" mid="3"?>lilacs<?oxy_comment_end?>
        flower reliably every year<?oxy_comment_end mid="3"?>
```

Managing Comments

A comment is marked in the **Author** mode with a background that is configured for each user name.

Section 3: Sample Section

And above all, remember that many flower gardens fail because they just don't get enough of your attention.

Drag and drop, cut, and copy operations are available on both CALS and HTML ▶ Docbook⁴ tables.

The built-in ▶ Docbook⁴ support in ▶ Oxygen⁴ includes a large set of operations and functionality. However, there are situations in which you must extend this set to match particular requirements.

Commented [Mary]:
 Let's add information about fertilizers.

Commented [John]: We should also say that the content of a table row or column can be deleted by selecting it and press DEL or Backspace.

Figure 60: Comments in Author Mode

You can manage comments using the following actions:



Add Comment...

Allows you to insert a comment at the cursor position or on a specific selection of content. The action is available in the Author toolbar.



Edit Comment...

Allows you to change an existing content. The action is available both in the Author toolbar and the contextual menu.



Remove Comment(s)...

Removes the comment at the cursor position or all comments found in the selected content. The action is available in the Author contextual menu, **Review** sub-menu.

Managing Highlights



Use the **Highlight** tool to mark the text in your document using different colours.



You can find the **Highlight** option on the main toolbar, in the **Edit > Review** menu, or in the contextual menu of a document, in the **Review** st of options.

What do you want to do?

- *Mark selected text;*
- *Mark fragments of the document you are editing;*
- *Remove highlighting.*



Tip: In case the **Highlight** tool is not available on your toolbar, enable **Author Comments** in the contextual menu of the toolbar.



Note: Oxygen XML Editor keeps the highlighting of a document between working sessions.

To watch our video demonstration about using the **Highlight** tool, go to http://oxygenxml.com/demo/Highlight_Tool.html.

Mark Selected Text

To mark the text you select in a document:

1. Select the text you want to highlight.



Note: To mark more than one part of the document you are editing, press and hold **Ctrl (Meta on Mac OS)** and using you cursor select the parts you want to highlight.

2. Click the small arrow next to the **Highlight** icon and select the colour that you want to use for highlighting.
The selected text is highlighted.

3. Click the **Highlight** icon to exit the highlighting mode.

Mark Document Fragments

To mark fragments in a document, follow these steps:

1. Click the  **Highlight** icon on the toolbar.
The highlighting mode is on. The cursor changes to a dedicated symbol that has the same color with the one set in the **Highlight** palette.
2. Select the text you want to highlight with your cursor.
3. To highlight different fragments using multiple colors, click the small arrow next to the  **Highlight** icon, choose the colour that you want to use for highlighting, and repeat step 2.
The fragments are highlighted.
4. To exist the highlighting mode, press **Esc** on your keyboard, click the  **Highlight** icon, or start editing the document.

Remove Highlighting from the Entire Document or Part of It.

To remove highlighting from the document you are editing, follow these steps:

1. Either select the text you want to remove highlighting from using your cursor, or press **Ctrl A (Command A on OS X)** in case you want to select all of the text.
2. Click the small arrow next to the  **Highlight** icon and select **No color (erase)**, or right click the highlighted content and select **Remove highlight(s)**.
The highlighting is removed.
3. Click the **Highlight** icon to exit the highlighting mode.

Author Callouts

A *callout* is a vertical stripe, with a balloon-like look, that Oxygen XML Editor displays in the right side of the editing area. Callouts are decorated with a colored border and also have a colored background. A horizontal line, which has the same color as the border, connects text fragments with their corresponding callouts. Oxygen XML Editor assigns an individual color for the callouts depending on the user who is editing the document. To customize the list of these colors, [open the Preferences dialog box](#) and go to **Editor > Edit Modes > Author > Review**. You are able to add, edit, or remove colors in this list. You can choose to use the same color for any user who modifies the content or inserts a comment. To do this, select the **fixed** option and choose a color from the color box. Once you set a fixed color for a user you are able to edit it. Press the color box and select a different color from the **Choose color** dialog box.

Oxygen XML Editor uses callouts to provide an enhanced view of the changes you, or other authors make to a document. They hold specific information depending on their type. In addition, Oxygen XML Editor uses callouts to display *comments* that you associate with fragments of the document you are editing. For more information about editing comments, go to [Managing Comments](#). To enable callouts, [open the Preferences dialog box](#) and go to **Editor > Edit Modes > Author > Review > Callouts**. Enable the following options:

- **Comments** - Oxygen XML Editor displays comment callouts when you insert a comment. You can use two types of comments in Oxygen XML Editor:
 - Author review comments: comments that you associate with specific fragments of text.
 - Change comments: comments that you add in an already existing insertion or deletion callout.

By default, the fragment of text that you comment is highlighted and a horizontal line connects it with the comment callout. A comment callout contains the name of the author who inserts the callout and the comment itself. To customize how comments are displayed, [open the Preferences dialog box](#), go to **Editor > Edit Modes > Author > Review > Callouts**, and enable **Show review time**.

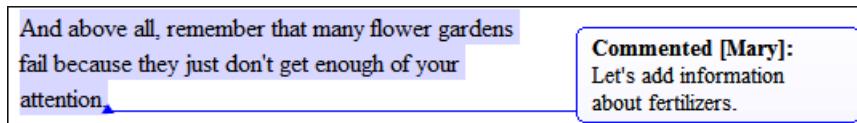


Figure 61: Comment Callouts

- **Track Changes deletions** - Oxygen XML Editor displays deletion callouts when you delete a fragment of text. By default, a deletion callout contains the type of callout (*Deleted*) and the name of the author that makes the deletion. You are able to customize the content of a deletion callout to display the date and time of the deletion and the deleted fragment itself. To do this, [open the Preferences dialog box](#), go to **Editor > Edit Modes > Author > Review > Callouts**, and enable **Show review time** and **Show deleted content in callout**.

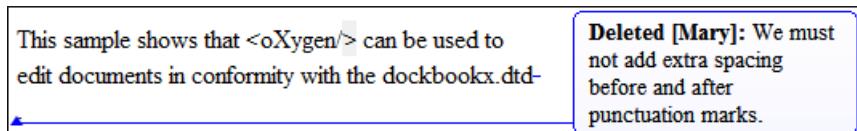


Figure 62: Deletion Callouts

- **Track Changes insertions** - Oxygen XML Editor displays insertion callouts when you insert a fragment of text. By default, an insertion callout contains the type of callout (*Inserted*) and the name of the author that makes the insertion. You are able to customize the content of an insertion callout to contain the date and time of the insertion and the inserted fragment itself. [Open the Preferences dialog box](#), go to **Editor > Edit Modes > Author > Review > Callouts**, and enable **Show review time** and **Show inserted content in callout**.

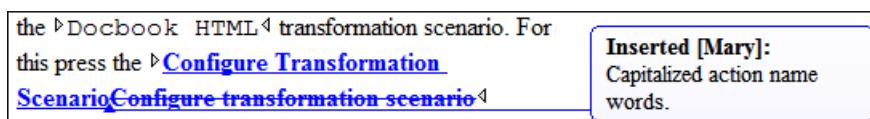


Figure 63: Insertion Callouts

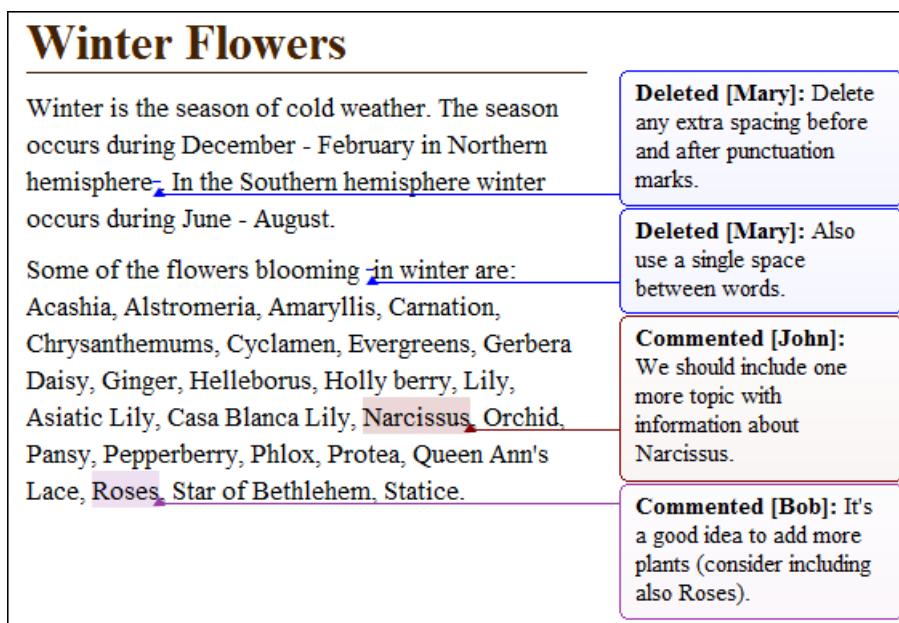


Figure 64: Multiple Authors Callouts

- Note:** Oxygen XML Editor displays callouts only if **View All Changes/Comments** or **View Only Changes/Comments** by is selected. Oxygen XML Editor does not display callouts in **View Final** and **View Original** modes.

To select a callout, either click the callout or its source. Selected callouts have a more intense background and a bold border. The connecting line between the source and the callout is also rendered in bold font. If you select a fragment of text which is associated with one or more callouts, the callouts are highlighted.

Important: The callouts are displayed in the right side of the editing area. However, in some cases, the text you are editing can span into the callouts area. For example, this situation can appear for callouts associated with wide images or space-preserve elements (like `codeblock` in DITA or `programlisting` in DocBook) which contain long fragments. To help you view the text under the covered area, Oxygen XML Editor applies transparency to these callouts. When the caret is located under a callout, the transparency is enhanced, allowing you to both edit the covered content and access the contextual menu of the editing area.

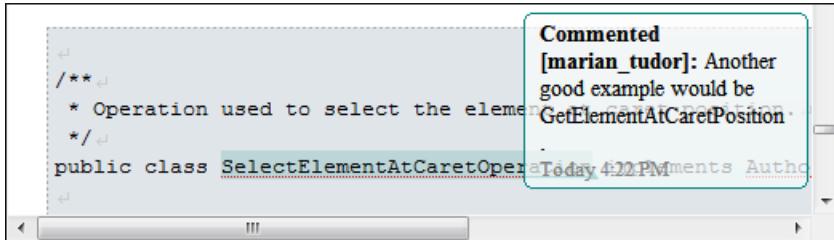


Figure 65: Transparent Callout

Note: Oxygen XML Editor does not display callouts located in folded areas of the edited document.

The following actions are available in the contextual menu of an insertion, or deletion callout:

- **Accept Change** - Select this option to accept the changes you or other authors make to a document.
- **Reject Change** - Select this option to reject the changes you or other authors make to a document.
- **Comment Change** - Select this option to comment an existing change in your document. You are also able to add a comment to a change from the **Comment Change** button available on the **Review** toolbar.
- **Edit Reference** - If the fragment that contains callouts is a reference, use this option to go to the reference and edit the callout.
- **Callouts Options** - Select this option to open the *preferences page* of the callouts.

The following options are available in the contextual menu of the comment callouts:

- **Edit Comment** - Select this option to modify the content of a comment callout;
- **Note:** The text area is disabled if you are not the author which inserted the comment.
- **Remove Comment** - Select this option to remove a comment callout.
- **Edit Reference** - If the fragment that contains callouts is a reference, use this option to go to the reference and edit the callout.
- **Callouts Options** - Select this option to open the Callouts *preferences page*.

When you print a document from Oxygen XML Editor, all callouts you, or other authors added to the document are printed. For a preview of the document and its callouts, go to **File > Print preview...**

To watch our video demonstration about the Callouts support, go to <http://oxygenvml.com/demo/CalloutsSupport.html>.

The Review View

The **Review** view is a framework-independent panel, available both for built-in, and custom XML document frameworks. It is designed to offer an enhanced way of monitoring all the changes that you make to a document. This means you are able to view and control highlighted, commented, inserted, and deleted content, or even changes made to attributes, using a single view.

The **Review** view is useful when you are working with documents that contain large quantities of edits. The edits are presented in a compact form, in the order they appear in the document. Each edit is marked with a type-specific icon.

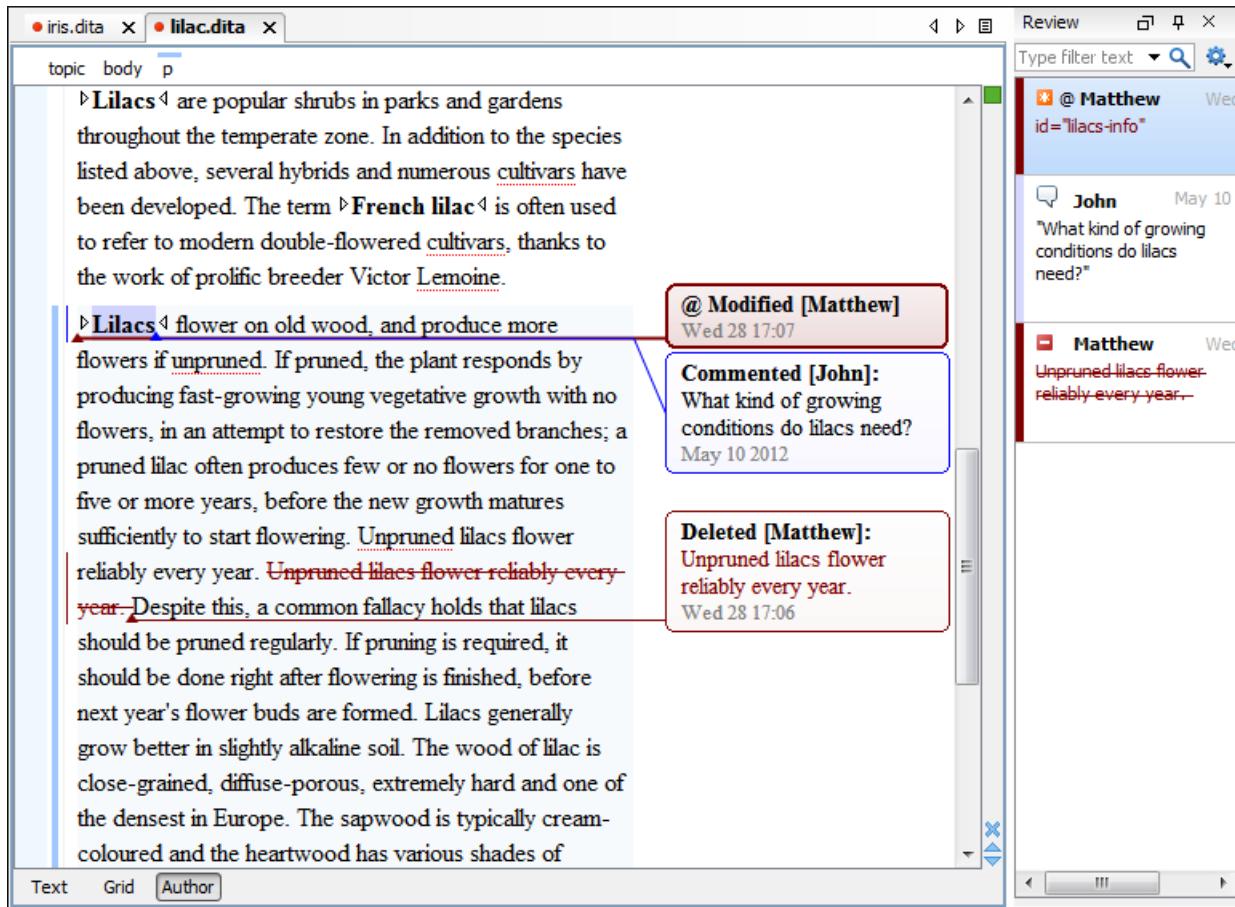


Figure 66: The Review View

To activate the **Review** view, do one of the following:

- click the **Manage reviews** button on the **Review** toolbar
- right click in a document and from the contextual menu go to **Review, Manage reviews**
- go to **Window > Show View > Review**

This view and the editing area are synchronized. When you select an edit listed in the **Review** view, its corresponding fragment of text is highlighted in the editing area and the reverse is also true. For example, when you place the caret inside an area of text marked as inserted, its corresponding edit is selected in the list.

The upper part of the view contains a filtering area which allows you to search for specific edits. Use the small arrow symbol from the right side of the search field to display the search history. The **Settings** button allows you to:

- **Show highlights** - controls whether the **Review** view displays the highlighting in your document.
- **Show comments** - controls whether the **Review** view displays the comments in the document you are editing.
- **Show track changes** - controls whether the **Review** view displays the inserted and deleted content in your document.
- **Show review time** - displays the time when the edits from the **Review** view were made.

The following actions are available when you hover the edits in the **Review** view, using the cursor:

Remove

Action available for highlights and comments presented in the **Review** view. Use this action to remove these highlights or comments from your document;

Accept

Action available for inserted and deleted content presented in the **Review** view. Use this action to accept the changes in your document;

Reject

Action available for inserted and deleted content presented in the **Review** view. Use this action to reject the changes in your document.

Depending on the type of an edit, the following actions are available in its contextual menu in the **Review** view:

Show comment

This option is available in the contextual menu of changes not made by you and of any comment listed in the **Review** view. Use this option to view a comment in the **Show comment** dialog.

Edit comment

This option is available in the contextual menu of your comments, listed in the **Review** view. Use this action to start editing the comment.

Remove comment

This option is available in the contextual menu of a comment listed in the **Review** view. Use this action to remove the selected comment.

Show only reviews by

This option is available in the contextual menu of any edit listed in the **Review** view. Use this action to keep visible only the edits of a certain author in the view.

Remove all comments

This option is available in the contextual menu of any comment listed in the **Review** view. Use this action to remove all the comments that appear in the edited document.

Change color

Opens a palette that allows you to choose a new color for the highlighted content.

Remove highlight

Removes the selected highlighting.

Remove highlights with the same color

Removes all the highlighting with the same color from the entire document.

Remove all highlights

Clears all the highlighting in your document.

Accept change

Accepts the selected change.

Reject change

Rejects the selected change.

Comment change

This option is available in the contextual menu of an insertion or deletion that you made. Use this option to open the **Edit comment** dialog and comment the change you made.

Accept all changes

Accepts all the changes made to a document.

Reject all changes

Rejects all the changes made to a document.

To watch our video demonstration about the **Review** view, go to http://oxygentools.com/demo/Review_Panel.html.

Profiling / Conditional Text

Conditional text is a way to mark blocks of text meant to appear in some renditions of the document, but not in others. It differs from one variant of the document to another, while unconditional text appear in all document versions.

For instance you can mark a section of a document to be included in the manual designated for the *expert* users, other for the *novice* users manual while unmarked sections are included in any rendition.

You can use conditional text when you develop documentation for:

- A series of similar products
- Different releases of a product
- Various audiences

The benefits of using conditional text include reduced effort for updating and translating your content and an easy way to customize the output for various audiences.

Oxygen XML Editor comes with a preconfigured set of profiling attribute values for some of the most popular document types. These attributes can be redefined to match your specific needs. Also, you can define your own profiling attributes for a custom document type.

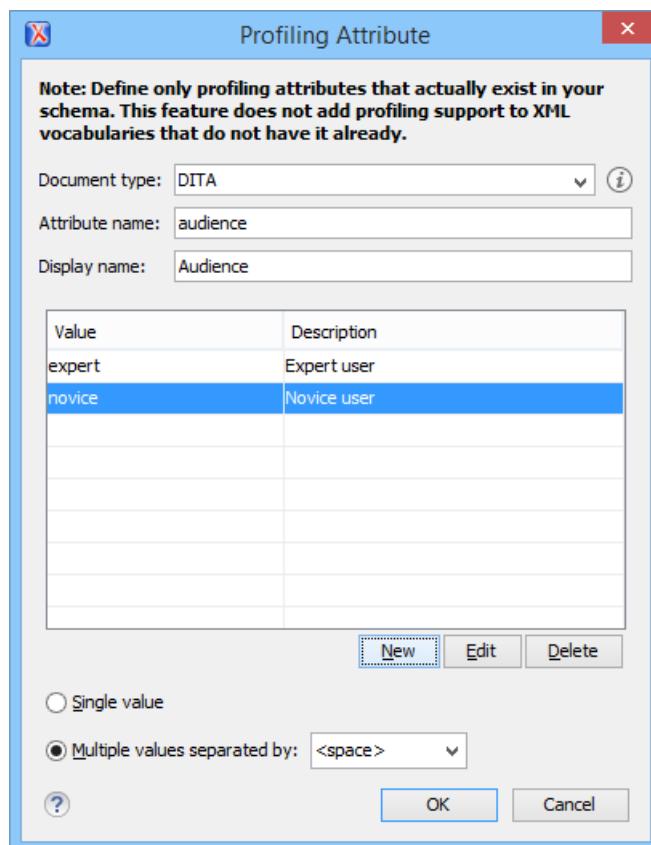
Create Profiling Attributes

 **Note:** To ensure the validity of the document, the attribute must already be defined in the document DTD or schema before referencing it here.

To create custom profiling attributes for a specific document type, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Profiling/Conditional Text**.
2. In the **Profiling Attributes** area, press the **New** button.

The **Profiling Attribute** dialog box is opened.



3. Fill-in the dialog box as follows:

- a) Choose the **Document type** on which the profiling attribute is applied. * and ? can be used as wildcards, while ,(comma character) can be used to specify more patterns. For example use *DITA** to match any document type name that starts with *DITA*.
- b) Specify the **Attribute name**.
- c) Specify a **Display name**. This field is optional, being used only as a descriptive rendering in profiling dialog boxes.

- d) Use the **New**, **Edit**, **Delete** buttons to add, edit, and delete possible values of the attribute. You can also specify and optional description for each attribute value.
- e) Choose whether the attribute accepts a **Single value** or **Multiple values separated by** a delimiter (*space, comma, semicolon*, or a custom one). A custom delimiter must be supported by the specified document type. For example, the DITA document type only accepts spaces as delimiters for attribute values.

4. Click **OK**.

5. Click **Apply** to save the profiling attribute.

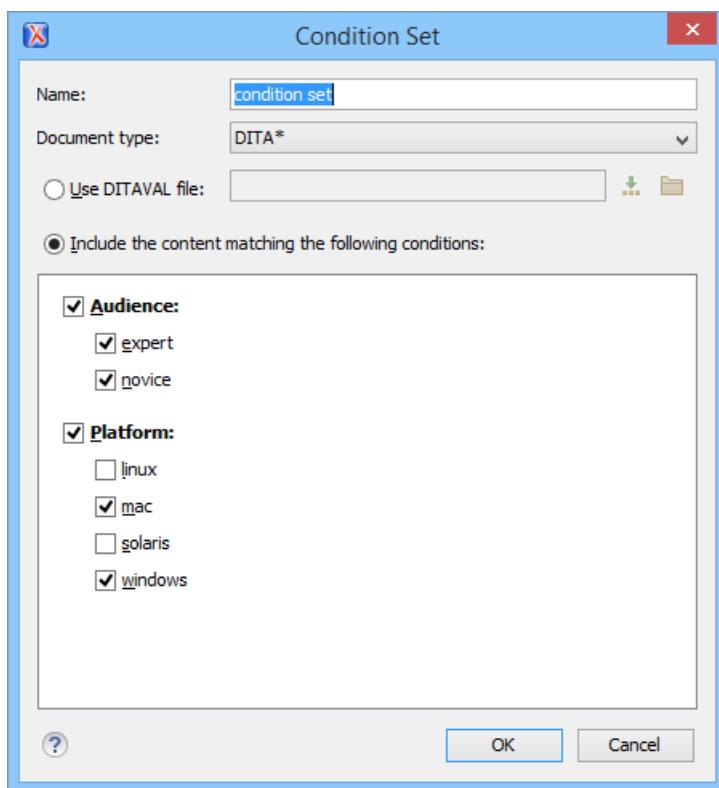
Create Profiling Condition Sets

Several profiling attributes can be aggregated into a profiling condition set that allow you to apply more complex filters on the document content. A Profiling Condition Set is a very powerful and convenient tool used to preview the content that goes into the published output. For example, an installation manual available both in Windows and Linux variants can be profiled to highlight only the Linux procedures for more advanced users.

To create a new profiling condition set:

1. [Open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Profiling/Conditional Text**.
2. In the **Profiling Condition Sets** area, press the **New** button.

The **Condition Set** dialog box is opened:



3. Fill-in the dialog box as follows:

- a) Type the condition set **Name**.

If you want the Profiling Condition Set to reference a DITAVAL file, enable the **Use DITAVAL file** option and select the DITAVAL file from your disk.

- b) Choose the **Document type** for which you have previously defined profiling attributes.

After choosing a document type, all profiling attributes and their possible values are listed in the central area of the dialog box.

- c) Define the combination of attribute values by selecting the appropriate checkboxes in the **Include the content matching the following conditions** section.

4. Click **OK**.

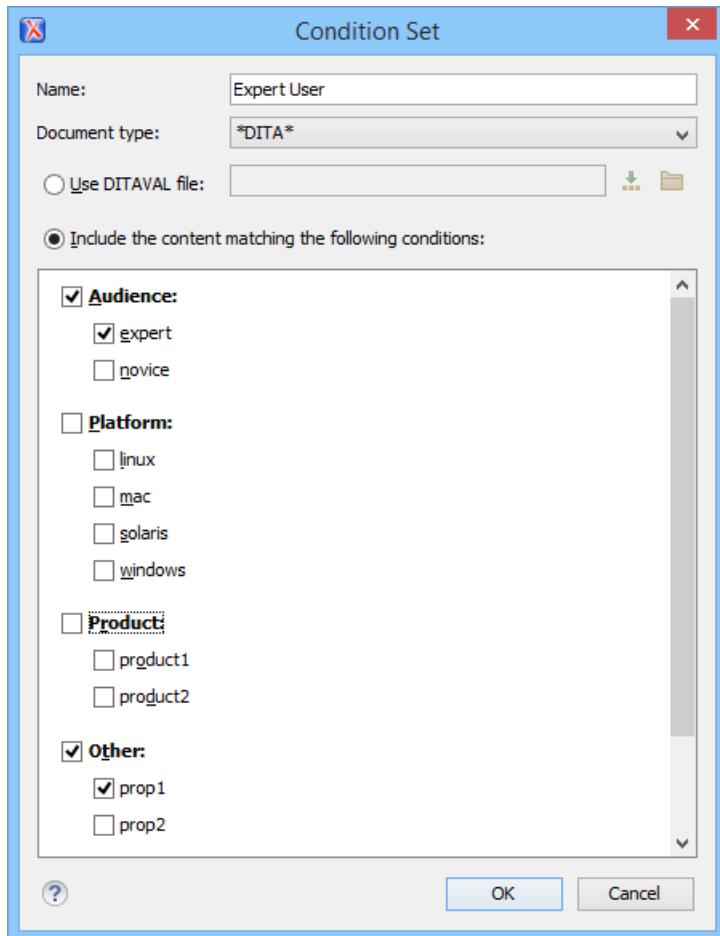
5. Click **Apply** to save the condition set. All saved profiling condition sets are available in the **Profiling / Conditional Text toolbar drop-down menu.**

Apply Profiling Condition Sets

All defined Profiling Condition Sets are available as shortcuts in the Profiling / Conditional Text menu. Just click on a menu entry to apply the condition set. The filtered content is grayed-out in Author editor, Outline view and DITA Maps Manager view. An element is filtered-out when one of its attributes is part of the condition set and its value does not match any of the value covered by the condition set. As an example, let us suppose that you have the following document:

Spray painting	
Short Description: When paint is applied using a spray nozzle, it is referred to as spray painting.	
Context:	
<p>▼ The garage is a good place to spray paint.</p>	
Step 1 Move the car out of the garage to avoid getting paint on it. Audience [novice]	
Step 2 Place newspaper, cardboard, or a drop-cloth on the garage floor. Audience [expert]	
Step 3 Place the object to be painted on the covered area. Audience [expert] Other [prop2]	
Step 4 Follow the directions on the paint can to paint the object. Audience [expert] Other [prop1]	
Step 5 Let the paint dry thoroughly before you move the object. Audience [novice] Other [prop1]	

If you apply the following condition set it means that you want to filter-out the content written for non-expert audience and having the *Other* attribute value different than *prop1*.



And this is how the document looks like after you apply the *Expert user* condition set:

Spray painting

Short Description: When paint is applied using a spray nozzle, it is referred to as spray painting.

Context:

▼ The garage is a good place to spray paint.

Step 1
Move the car out of the garage to avoid getting paint on it. Audience [novice]

Step 2
Place newspaper, cardboard, or a drop-cloth on the garage floor. Audience [expert]

Step 3
Place the object to be painted on the covered area. Audience [expert] Other [prop2]

Step 4
Follow the directions on the paint can to paint the object. Audience [expert] Other [prop1]

Step 5
Let the paint dry thoroughly before you move the object. Audience [novice] Other [prop1]

Apply Profiling Attributes

Profiling attributes are applied on element nodes.

You can apply profiling attributes on a text fragment, on a single element, or on multiple elements in the same time. To profile a fragment from your document, select the fragment in the **Author** mode and follow these steps.



Note: If there is no selection in your document, the profiling attributes are applied on the element at caret position.

1. Invoke the **Edit Profiling Attributes...** action from the contextual menu.

The displayed dialog box shows all profiling attributes and their values, as defined on the document type of the edited content. The checkboxes corresponding with the values already set in the profiled fragment are enabled.

2. In the **Edit Profiling Attributes** dialog box, enable the checkboxes corresponding to the attribute values you want to apply on the document fragment. The profiling attributes having different values set in the elements of the profiled fragment are marked with a gray background and they are disabled by default. You can change the values of these attributes by choosing the **Change Now** option associated with all attributes.
3. Click **OK** to finish the profiling configuration.

The attributes and attributes values selected in the **Edit Profiling Attributes** dialog box are set on the elements contained in the profiled fragment.

If you select only a fragment of an element's content, this fragment is wrapped in phrase-type elements on which the profiling attributes are set. Oxygen XML Editor comes with predefined support for DITA and DocBook. For more developer-level customization options, see the [Customize Profiling Conditions](#) topic.

If **Show Profiling Attributes** option (available in the **Profiling / Conditional Text toolbar menu**) is set, a light green border is painted around profiled text, in the **Author** mode. Also, all profiling attributes set on the current element are listed at the end of the highlighted block and in its tooltip message. To edit the attributes of a profiled fragment, click one of the listed attributes. A form control pops up and allows you to add or remove attributes using their checkboxes.

Profiling / Conditional Text Menu

The **Profiling / Conditional Text** toolbar menu groups the following actions:

Show Profiling Colors and Styles

Enable this option to turn on conditional styling.

Show Profiling Attributes

Enable this option to turn on conditional text markers. They are displayed at the end of conditional text blocks, as a list of attribute name and their currently set values.

Show Excluded Content

Controls if the content filtered out by a particular condition set is hidden or greyed-out in the editor area and in the **Outline** and **DITA Maps Manager** views. When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.



Note: To remind you that document content is hidden, Oxygen XML Editor displays labels showing the currently applied condition set. These labels are displayed in the Author editing area, the **Outline** view and **DITA Maps Manager** view. Right click any of the labels to quickly access the **Show Excluded Content** action.

List of all profiling condition sets that match the current document type

Click on a condition set entry to activate it.

Profiling Settings...

Link to the [profiling options](#) preference pages, where you can manage profiling attributes, profiling conditions sets, as well as profiling styles and colors options.

All these settings are associated with the current project, being restored the next time you open it. For a new project all Profiling/Conditional Text menu actions states are reset to their default values.

Apply Profiling Colors and Styles

Applying profiling colors and styles allows you to customize the Author editing area to mark profiled content so you can instantly spot different variants of the output.

Mowing equipment needs regular checks and maintenance. Monthly, you should:

- Refill the oil:
 - Remove the oil fill cap;
 - Pour new oil gradually. Regularly check the dipstick to see if the oil level reached the maximum mark;
- Sharpen the blades:
 - Clamp the blade to a vice or to the edge of a solid surface;
 - Using an electric grinder or audience [ExpertUser] a file, grind the length of the blade until it is sharp;
- Check the electric cable for any signs of wear. Replace it if worn; product [Electric]
- Clean the mower's underside for debris; product [Electric, Gasoline]
- Inspect the general state of the mower. Use a ratchet to tighten any loose bolts;
- Lubricate the gears of the manual lawn mower; product [Manual]

Choosing the right style for a specific profiling attribute is a matter of personal taste, but you should keep in mind that:

- If the same block of text is profiled with two or more profiling attributes, their associated styles combine. Depending on the styling, this might result in an excessively styled content that may prove difficult to read or work with.
- Profile only differences. There is no need to profile common content, since excessive profiling can visually pollute the document.
- A mnemonic associated with a style will help you spot instantly different types of content.

To set colors and styles to profiling attribute values:

- Enable the **Show Profiling Colors and Styles** option from the *Profiling / Conditional Text toolbar drop-down menu*.
- Go to **Profiling Settings** from the *Profiling / Conditional Text toolbar drop-down menu*. This is a shortcut to the *Profiling/Conditional Text options page*. Select the *Colors and Styles options page*.
- Set a style to a profiling attribute value.

Note that the styling is now applied in the Author editor, the Outline view and DITA Maps Manager view. Also, to help you identify more easily the profiling you want to apply in the current context, the styling is applied in the **Edit Profiling Attributes** dialog box and in the inline form control that allows you to quickly set the profiling attributes.

Smart Paste Support

You can paste content from various sources, such as web pages and office-type documents, and paste it into DITA, TEI, DocBook, and XHTML documents. Oxygen XML Editor keeps the original text styling (like bold, italics) and formatting (like lists, tables, paragraphs), and helps you make the resulting document valid.

You can paste content from the following:

- Office applications (**Microsoft Word** and **Microsoft Excel**, **OpenOffice.org Writer** and **OpenOffice.org Calc**).
- Web browsers.
- The Oxygen XML Editor **Data Source Explorer** view (where resources are available from WebDAV or CMS servers).

The following document types have smart paste support:

- [DITA](#)
- [DocBook 4](#)
- [DocBook 5](#)
- [TEI 4](#)
- [TEI 5](#)
- [XHTML](#)
- [JATS](#)

The styles and general layout of the pasted content are transformed to the equivalent XML markup of the target document type.

Tables pasted in a DocBook file are automatically converted to CALS. If you want to overwrite this behaviour and instruct Oxygen XML Editor to convert them to HTML tables, set the `docbook.html.table` parameter to 1. You can find this parameter in:

- `[OXYGEN_DIR]/frameworks/docbook/resources/xhtml2db5Driver.xsl` stylesheet, for DocBook 5
- `[OXYGEN_DIR]/frameworks\docbook\resources\xhtml2db4Driver.xsl` stylesheet, for DocBook 4

You can disable smart paste by deselecting [Convert external content on paste](#) in the **Schema Aware** preferences.

If you paste the content in a location where the resulting XML would not be valid, Oxygen XML Editor will attempt to place it in a valid location, and may prompt you with one or more choices for where to place it.

You can disable this location selection feature by deselecting [Smart paste and drag and drop](#) option, available in the **Schema Aware** preferences.

To watch our video demonstration about the Smart Paste support, go to

http://oxygenvml.com/demo/Smart_Paste_Copy_Paste_from_Web_Office_Documents_to_DITA_DocBook_TEI_XHTML_Documents.html.

Bidirectional Text Support in Author Mode

Oxygen XML Editor offers support for languages that require right to left scripts. This means that authors editing documents in the **Author** mode are able to create and edit XML content in Arabic, Hebrew, Persian and others. To achieve this, Oxygen XML Editor implements the [Unicode Bidirectional Algorithm](#) as specified by the Unicode consortium. The text arrangement is similar to what you get in a modern HTML browser. The final text layout is rendered according with the directional CSS properties matching the XML elements and the Unicode directional formatting codes.

If bidirectional text (such as Arabic or Hebrew languages), certain Asian languages (such as Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Sinhala, Thai, Khmer), or other special characters (such as combining characters) are detected in a document, Oxygen XML Editor displays a **Special Characters Detected** dialog box that prompts you to **Enable** or **Disable** support for these special characters.

You can also configure this support in the **Support for Special Characters** section of the **Open/Save** preferences page. To enable or disable this support, [open the Preferences dialog box](#) and go to **Editor > Open/Save**.

 **Note:** Disabling this support may affect text rendering, cursor positioning and navigation, as well as text selection and management operations. If you need to open [very large documents](#), the bidirectional editing support can also be disabled to enhance performance while editing.

To watch our video demonstration about the bidirectional text support in the **Author** mode, go to http://oxygenvml.com/demo/BIDI_Support.html.

Controlling the Text Direction Using XML Markup

Oxygen XML Editor Supports the following CSS properties:

Table 5: CSS Properties Controlling Text Direction

direction	Specifies the writing direction of the text. The possible values are <code>ltr</code> (the text direction is left to right), <code>rtl</code> (the text direction is right to left, and <code>inherit</code> (specifies whether the value of the <code>direction</code> property is inherited from the parent element).
unicodeBidi	Used with the <code>direction</code> property, sets or returns whether the text is overridden to support multiple languages in the same document. The possible values of this property are <code>bidi-override</code> (creates an additional level of embedding and forces all strong characters to the direction specified in the <code>direction</code>), <code>embed</code> (creates an additional level of embedding), <code>normal</code> (does not use an additional level of embedding), and <code>inherit</code> (the value of the <code>unicodeBidi</code> property is inherited from parent element).

For instance, to declare an element as being Right to Left, you could use a stylesheet like the one below:

XML File:

```
<article>
  <myRTLpara>RIGHT TO LEFT TEXT</myRTLPara>
</article>
```

Associated CSS File:

```
myRTLpara{
  direction:rtl;
  unicode-bidi:embed;
}
```

Oxygen XML Editor recognizes the `dir` attribute on any XML document. The supported values are:

ltr	The text from the current element is Left to Right, embedded.
rtl	The text from the current element is Right to Left, embedded.
lro	The text from the current element is Left to Right, embedded.
rlo	The text from the current element is Right to Left, embedded.

The following XML document types make use of the `dir` attribute with the above values:

- DITA
- DocBook
- TEI
- XHTML



Note: When the inline element tags are visible, the text in the line is arranged according to the BIDI algorithm after replacing the tags symbols with Object Replacement Characters. This makes it possible to get a different text arrangement when viewing a document in the **No Tags** mode versus viewing it in the **Full Tags** mode.

Controlling the Text Direction Using the Unicode Direction Formatting Codes

These Unicode Direction Formatting Codes codes can be embedded in the edited text, specifying a text direction and embedding. However, it is not recommended to use them in XML as they are zero width characters, making it hard to debug the text arrangement.

Table 6: Directional Formatting Codes

U+202A (LRE)	LEFT-TO-RIGHT EMBEDDING	Treats the following text as embedded left-to-right.
U+202B (RLE)	RIGHT-TO-LEFT EMBEDDING	Treats the following text as embedded right to left.
U+202D (LRO)	LEFT-TO-RIGHT OVERRIDE	Forces the following characters to be treated as strong left-to-right characters.
U+202E (RLO)	RIGHT-TO-LEFT OVERRIDE	Forces the following characters to be treated as strong right-to-left characters.
U+202C (PDF)	POP DIRECTIONAL FORMATTING CODE	Restores the bidirectional state to what it was before the last LRE, RLE, RLO, or LRO.
U+200E (LRM)	LEFT-TO-RIGHT MARK	Left-to-right strong zero-width character.
U+200F (RLM)	RIGHT-TO-LEFT MARK	Right-to-left strong zero-width character.

To insert Unicode Direction Formatting Codes, use the [Character Map](#) dialog box. To easily find such a code, you can either enter directly the hexadecimal value, or use the **Details** tab to enter the codes name.

Oxygen XML Editor offers the support for bi-directional text in all the side views (**Outline** view, **Attributes** view and so on) and text fields.

Chapter

7

Editing Documents

Topics:

- [*Working with Unicode*](#)
- [*Creating, Opening, and Closing Documents*](#)
- [*Grouping Documents in XML Projects*](#)
- [*Editing XML Documents*](#)
- [*Editing XSLT Stylesheets*](#)
- [*Editing Ant Build Files*](#)
- [*Editing XML Schemas*](#)
- [*Editing XQuery Documents*](#)
- [*Editing WSDL Documents*](#)
- [*Editing CSS Stylesheets*](#)
- [*Editing LESS CSS Stylesheets*](#)
- [*Editing Relax NG Schemas*](#)
- [*Editing NVDL Schemas*](#)
- [*Editing JSON Documents*](#)
- [*Editing StratML Documents*](#)
- [*Editing JavaScript Documents*](#)
- [*Editing XProc Scripts*](#)
- [*Editing Schematron Schemas*](#)
- [*Editing Schematron Quick Fixes*](#)
- [*Editing SVG Documents*](#)
- [*Editing XHTML Documents*](#)
- [*Spell Checking*](#)
- [*AutoCorrect Misspelled Words*](#)
- [*Editing Large Documents*](#)
- [*Scratch Buffer*](#)
- [*Handling Read-Only Files*](#)
- [*Editing Documents with Long Lines*](#)
- [*Associating a File Extension with Oxygen XML Editor*](#)

This chapter explains the editor types available in Oxygen XML Editor and how to work with them for editing different types of documents.

Working with Unicode

Unicode provides a unique number for every character, independent of the platform and language. Unicode is an internationally recognized standard, adopted by industry leaders. The Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646.

It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends. Incorporating Unicode into client-server or multi-tiered applications and websites offers significant cost savings over the use of legacy character sets.

As a modern XML Editor, Oxygen XML Editor provides support for the Unicode standard enabling your XML application to be targeted across multiple platforms, languages, and countries without re-engineering. Internally, the Oxygen XML Editor XML Editor uses 16bit characters covering the Unicode Character set.

As a Java application, Oxygen XML Editor comes with a default Java input method for typing characters with Unicode codes. However, the default input method does not cover all the Unicode codes, for example the codes for some accented characters or characters found in East Asian languages. Such characters can be inserted in the editor panel of Oxygen XML Editor either with [the Character Map dialog](#) available from menu **Edit > Insert from Character Map** or by installing a Java input method that supports the insertion of the needed characters. The [installation of a Java input method](#) depends on the platform on which Oxygen XML Editor runs (Windows, Mac OS X, Linux, etc) and is the same for any Java application.

 **Note:** Oxygen XML Editor may not be able to display characters that are not supported by the operating system (either not installed or unavailable).

 **Tip:** On windows, you can enable the support for CJK (Chinese, Japanese, Korean) languages from **Control Panel / Regional and Language Options / Languages / Install files for East Asian languages**.

Opening and Saving Unicode Documents

When loading documents, Oxygen XML Editor reads the document prolog to determine the specified encoding type. This encoding is then used to instruct the Java Encoder to load support for and to save the document using the specified code chart. When the encoding type cannot be determined, Oxygen XML Editor prompts and display the **Available Java Encodings** dialog box that provides a list of all encodings supported by the Java platform.

If the opened document contains an unsupported character, Oxygen XML Editor applies [the policy specified for handling such errors](#). If the policy is set to REPORT, Oxygen XML Editor displays an error dialog box with a message about the character not allowed by the encoding. If the policy is set to IGNORE, the character is removed from the document displayed in the editor panel. If the policy is set to REPLACE, the character is replaced with a standard replacement character for that encoding.

While in most cases you are using UTF-8, simply changing the encoding name causes the application to save the file using the new encoding.

When saving a document edited in the **Text**, **Grid**, or **Design** modes, if it contains characters not included in the encoding declared in the document prolog, Oxygen XML Editor detects the problem and signals it to the user. The user is responsible to resolve the conflict before saving the document.

When saving a document edited in the **Author** mode, all characters that fall outside the detected encoding will be automatically converted to hexadecimal character entities.

To edit documents written in Japanese or Chinese, change the font to one that supports the specific characters (a Unicode font). For the Windows platform, *Arial Unicode MS* or *MS Gothic* is recommended. Do not expect *WordPad* or *Notepad* to handle these encodings. Use *Internet Explorer* or *Word* to examine XML documents.

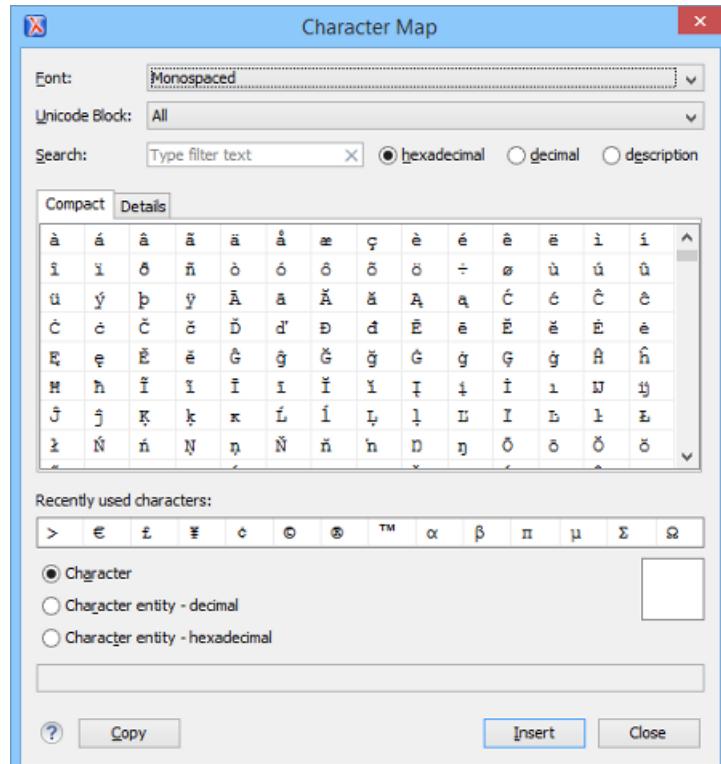
When a document with a UTF-16 encoding is edited and saved in Oxygen XML Editor, the saved document has a byte order mark (BOM) which specifies the byte order of the document content. The default byte order is platform-dependent. That means that a UTF-16 document created on a Windows platform (where the default byte order mark is *UnicodeLittle*)

has a different BOM than a UTF-16 document created on a Mac OS platform (where the byte order mark is *UnicodeBig*). The byte order and the BOM of an existing document are preserved when the document is edited and saved. This behavior can be changed in Oxygen XML Editor from the [Encoding preferences panel](#).

Inserting Symbols

You can insert symbols by using the **Edit > Insert from Character Map...** action. This displays the **Character Map** dialog box.

The Character Map Dialog Box



The **Character Map** dialog box allows you to visualize all characters available in a font, pick the character you need and insert it in the document you are editing. You can also access this dialog if you use the **Edit > Insert from Character Map...** action.

Use the **Font** box to choose the font for which you want to display characters. To see only a certain range of characters, use the **Unicode Block** box. This will filter the number of characters displayed, showing only a contiguous range of characters corresponding to the selected block. Unassigned characters are displayed as empty squares.

The available characters are listed in two tabs:

- **Compact** - matrix-like representation which displays only characters.
- **Details** - displays the available characters in a tabular format, presenting their decimal and hexadecimal value along with their description.

Use the **Search** field to search for a character by one of the following attributes:**decimal** value, **hexadecimal** value or description. Selecting the **description** option places the focus on the **Details** tab. In case you enter a character description in the **Search** field, the **description** option is selected automatically. The searching operation starts as soon as you start typing characters in the **Search** field.

Press the **Insert** button to insert the selected character in the current editor at caret position. You will see the character in the editor if [the editor font](#) is able to render it. The **Copy** button copies it to the clipboard without inserting it in the editor. You can see the name and range name of a character either at the bottom of the dialog, or when hovering the mouse cursor over the character.

The **Character Map** dialog cannot be used to insert Unicode characters in *the grid version of a document editor*.

Accordingly, the **Insert** button of the dialog will be disabled if the current document is edited in **Grid** mode.

Creating, Opening, and Closing Documents

This section explains the actions and wizards available for creating new files, opening existing files, and closing files.

Creating Documents

This section details the procedures available for creating new documents.

The New Document Wizard

Oxygen XML Editor supports a wide range of document types. The **New Document** wizard presents the default associations between a file extension and the type of editor that opens the file. To customize these default associations, *open the Preferences dialog box* and go to **File Types**.

1. To create a document in Oxygen XML Editor, either select **File > New > Ctrl N (Command N on OS X)**, or click the  **New** button on the toolbar.

Oxygen XML Editor displays the **New Document** wizard and groups the supported document types in multiple categories:

- **Recently used** - Contains the list of the most recently used files.
- **New Document** - Contains the list of all supported document types. This list includes XML, XSL, XML Schema, Document Type Definition, Relax NG Schema, XQuery, web Services Definition Language, Schematron Schema, CSS, Text, PHP, JavaScript, Java, C, C++, Batch, Shell, Properties, SQL, XML Catalog, and PERL.
- **Global templates** - contains the list of predefined templates as well as templates defined in the *Document Templates* preferences page.
- **Framework templates** - contains the list of templates defined in the *Document Type Association* preferences page, **Templates** tab.

2. Select a document type.

3. Click one of the following:

- **Customize** - action available only for XML, XML Schema, Schematron, and XSL documents. Depending on the document type, you can set different properties before you create the file.
- **Create** - uses default settings to create a file.

If you select **Create**, Oxygen XML Editor opens the new file in the editor view.

4. If you select **Customize**, Oxygen XML Editor opens the following dialog box. You can customize different options depending on the document type you select.

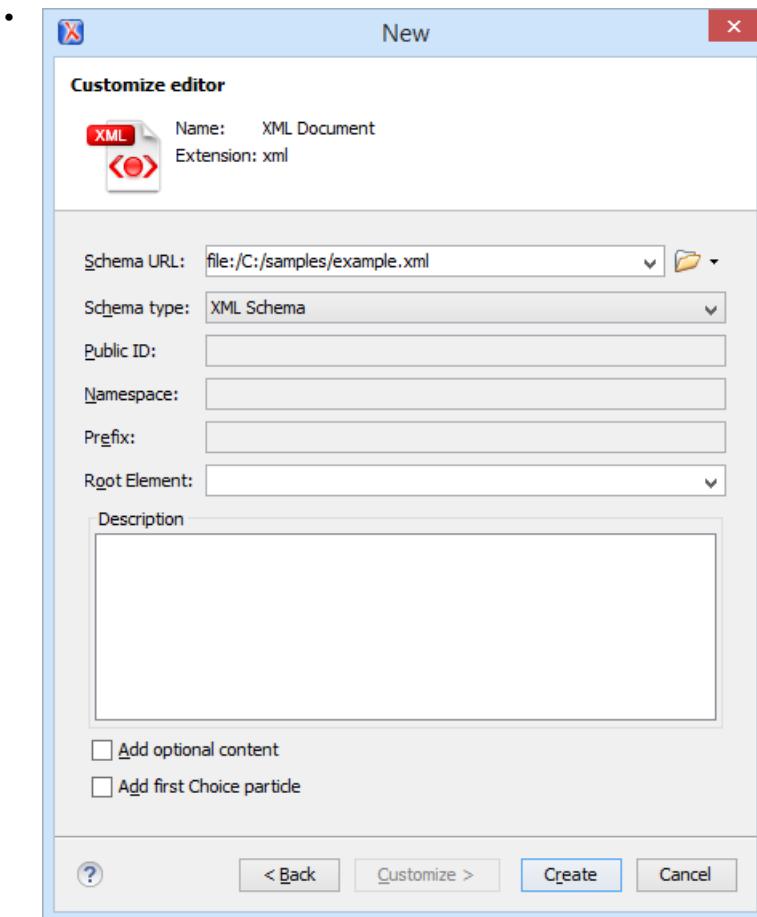


Figure 67: New XML Document Dialog Box

- **Schema URL** - Specifies the path to the schema file. When you select a file, Oxygen XML Editor analyzes its content and tries to fill the rest of the dialog box.
- **Schema type** - Allows you to select the schema type. The following options are available: XML Schema, DTD, RelaxNG XML syntax, RelaxNG compact syntax, and NVDL.
- **Public ID** - Specifies the PUBLIC identifier declared in the document prolog.
- **Namespace** - Specifies the document namespace.
- **Prefix** - Specifies the prefix for the namespace of the document root.
- **Root Element** - Populated with elements defined in the specified schema, enables selection of the element used as document root.
- **Description** - Shows a small description of the selected document root.
- **Add optional content** - If you select this option, the elements, and attributes defined in the XML Schema as optional, are generated in the skeleton XML document.
- **Add first Choice particle** - If you select this option, Oxygen XML Editor generates the first element of an `xs:choice` schema element in the skeleton XML document. Oxygen XML Editor creates this document in a new editor panel when you click **OK**.

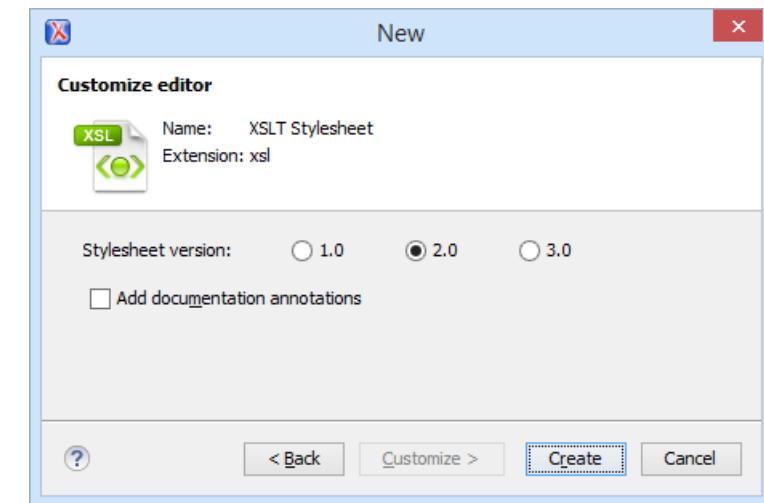


Figure 68: New XSL Document Dialog Box

- **Stylesheet version** - Allows you to select the Stylesheet version number. You can select from: 1.0, 2.0, and 3.0.
- **Add documentation annotations** - Enable this option to generate the stylesheet documentation.

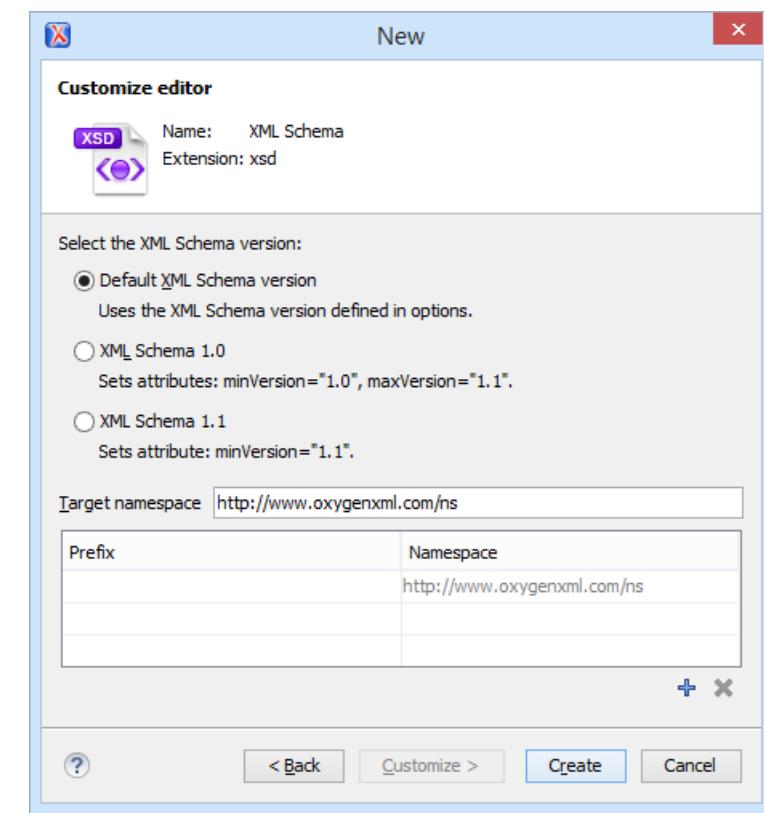


Figure 69: New XML Schema Document Dialog Box

- **Default XML Schema version** - Uses the XML Schema version defined in the **XML Schema** preferences page.
- **XML Schema 1.0** - Sets the `minVersion` attribute to `1.0` and the `maxVersion` attribute to `1.1`.
- **XML Schema 1.1** - Sets the `minVersion` attribute to `1.1`.
- **Target namespace** - specifies the schema target namespace.

- **Namespace prefix declaration table** - contains namespace prefix declarations. Table information can be managed using the **New** and **Delete** buttons.
- Tip:** For further details on how you can set the version of an XML Schema, go to [Setting the XML Schema Version](#).

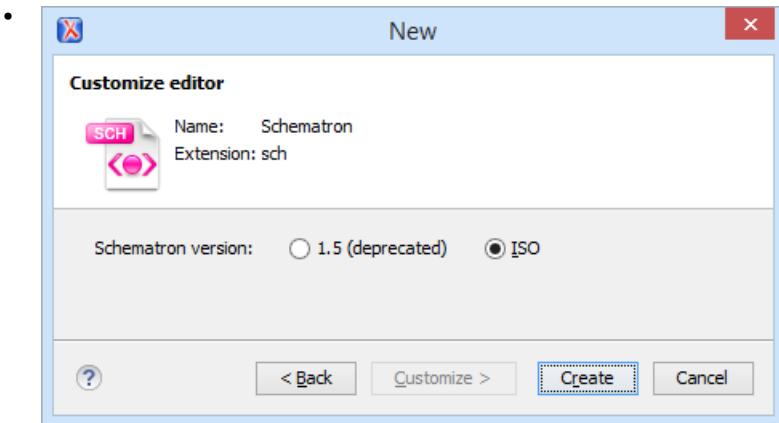


Figure 70: New Schematron Document Dialog Box

- **Schematron version** - Specifies the Schematron version. Possible options: 1.5 and ISO.
- Note:** Starting with version 16.0 of Oxygen XML Editor, the support for Schematron 1.5 is deprecated. It is recommended to use ISO Schematron instead.

To easily create a Schematron document the application offers two predefined document templates, ISO Schematron and Schematron 1.5.

5. Press **Create** to create the file.

Creating Documents Based on Templates

The [New wizard](#) enables you to select predefined templates or custom templates. Custom templates are created in previous sessions or by other users.

The list of templates presented in the dialog includes:

- Document Types templates - Templates supplied with the defined document types.
- User defined templates - You can add template files to the `templates` folder of the Oxygen XML Editor install directory. You can also specify another directory to use for templates. [Open the Preferences dialog box](#) and go to **Editor > Templates > Document Templates** to specify a custom templates folder.

1. Go to menu **File > New**.
2. Select a document type.
3. Press the **Finish** button.

The newly created document already contains the structure and content provided in the template.

Document Templates

Templates are documents that have a predefined structure. They provide the starting point from which you can build new documents rapidly, based on the same characteristics (file type, prolog, root element, existing content). Oxygen XML Editor offers a rich set of templates for a number of XML applications. You may also create your own templates and share them with others.

To configure or add templates, [open the Preferences dialog box](#) and go to **Editor > Templates > Document Templates**. You can also use [editor variables](#) in the template files' content and they will be expanded when the files are opened.

Saving Documents

You can save the document you are editing with one of the following actions:

- **File >  Save.**
- The ** Save** toolbar button. If the document was not saved yet it displays the **Save As** dialog.
- **File > Save As** - displays the **Save As** dialog, used either to name and save an open document to a file or to save an existing file with a new name.
- **File > Save To URL** - displays the **Save to URL** dialog, which can be used to save a file identified by its URL (defined by a protocol, host, resource path, and an optional port). Use the drop down action list to choose one of the available save actions:
 - ** Browse for local file** - Opens a local file browser dialog box allowing you to save the document locally.
 - ** Browse for remote file** - Displays the **Save to URL** dialog which allows you to save the document to a remote location (accessible through FTP, SFTP or WebDAV).
 - ** Browse for archived file** - Displays the **Archive Browser Dialog**, which allows you to save the document inside an archive.
 - ** Browse Data Source Explorer** - Opens the **Data Source Explorer** which allows you to browse the data sources defined in the [Data Sources preferences page](#).
- ** Tip:** You can get to the **Data Sources** preferences page, using the **Configure Database Sources** shortcut from the **Save to URL** dialog.
- ** Search for file** - Displays the [Open/Find Resources dialog](#).
- **File > Save All** - saves all open documents. If any document does not have a file, displays the **Save As** dialog.

Opening/Navigating Documents

To open a document in Oxygen XML Editor, do one of the following:

- Go to **File >  Open... (**Ctrl O (Command O on OS X)**)** or click the ** Open...** toolbar button to display the **Open** dialog box. The start folder of the **Open** dialog box can be either the last folder visited by this dialog box or the folder of the currently edited file. This can be [configured in the user preferences](#).
- Go to **File >  Open URL...** or click the ** Open URL...** toolbar button to display the **Open URL** dialog box, which allows you to access any resource identified through an URL (defined by a protocol, host, resource path, and an optional port). The following actions are available in the drop-down action list:
 - ** Browse for local file** - Opens a local file browser dialog box allowing you to select a local file.
 - ** Browse for remote file** - Displays [the Open using FTP/SFTP dialog box](#) that allows you to open a remotely stored document.
 - ** Browse for archived file** - Displays the **Archive Browser** dialog box that allows you to browse the content of an archive and choose a file to open in Oxygen XML Editor.
 - ** Browse Data Source Explorer** - Opens the **Data Source Explorer** that allows you to browse the data sources defined in the [Data Sources preferences page](#).
- ** Tip:** You can get to the **Data Sources** preferences page, using the **Configure Database Sources** shortcut from the **Open URL** dialog box.
- ** Search for file** - Displays the [Open/Find Resources dialog box](#).
- Click the ** Open/Find Resource** toolbar button to run the same action.
- Go to **File > Reload** to load the last saved file content. All unsaved modifications are lost.

- Go to **File > Reopen** to reopen one of the recently opened document files. The list containing recently opened files can be emptied by invoking the **Clear history** action.
- Select the **Open** action from the contextual menu of the **Project** view. This opens the selected file from the **Project** view.

The Open/Find Resource View

The **Open/Find Resource** view is designed to offer advanced search capabilities either by using a simple text search or by using the *Apache Lucene - Query Parser Syntax*. To open this view, go to **Window > Show View > Open/Find Resource**. The view is presented in the left side of the default Oxygen XML Editor layout, next to the **Project** and **DITA Maps Manager** views.

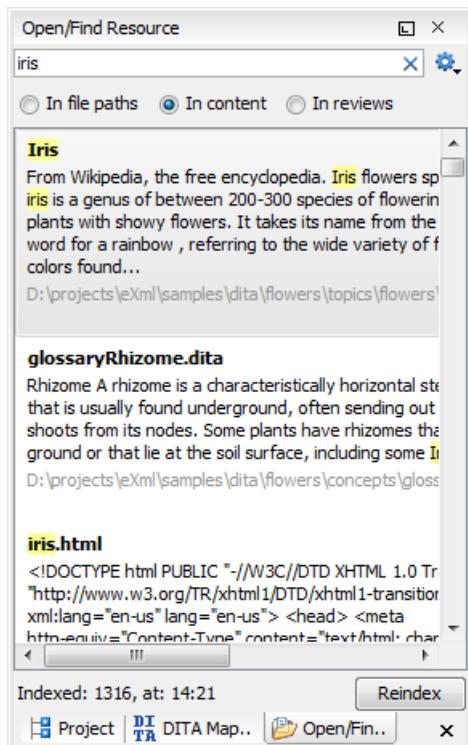


Figure 71: The Open/Find Resource View

You can use this view to find a file in the current Oxygen XML Editor project or in one of the DITA maps opened in *the DITA Maps Manager view* by typing only a few letters of the file name of a document or a fragment of the content you are searching for. The Open/Find Resource view also supports searching in document edits (comments, insertions, deletions, and highlighted content).



Note: Full support for searching in document edits is available only in the Enterprise edition of Oxygen XML Author and Oxygen XML Editor. The Professional edition offers support to search through a maximum of 10 edits.

Search results are presented instantly, after you finish typing the content you are searching for. The matching fragments of text are highlighted in the results list displayed in the view. When you open one of the documents from the results list, the matching fragments of text are highlighted in the editing area. To remove the highlighting from your document close the **Results view**. To display the search history, position the caret in the search field and press **Ctrl Down Arrow (Command Down Arrow on OS X)** or **Ctrl Up Arrow (Command Up Arrow on OS X)** on your keyboard. Pressing only the **Down Arrow** key moves the selection to the list of results.

A contextual menu available on each search result provides actions applicable to the document that contains it. These actions allow you to:

- Open the document in one of Oxygen XML Editor internal editors.

- Open the document in an external system application.
- Identify the document in the system file explorer.
- Copy the file's location.

The content of the resources used to search in is parsed from an index. The indexing is performed both automatically and on request.

-  **Note:** Automatic indexing is performed when you modify, add, or remove resources in the currently indexed project. In case the index was never initialized, the index is not updated on project changes.
-  **Note:** To improve performance, the indexing process skips the following set of common English words (the so-called *stop words*): *a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with*. This means that if you are searching for any of these words, **Open/Find Resource** view will not be able to match any of them. However, you can configure the list of *stop words* in the [Open/Find Resources Preferences Page](#).
-  **Note:** Searches are case insensitive. For example, if you search for *car* you get the same results as when you search for *Car*.
-  **Note:** Suffix searches are supported, both for searching in the content of your resources and in their name. For this, you can use wildcards. If you search **in content** for **.ing* you will find documents that contain the word *presenting*. If you search **in file paths** for **/samples/*.gif* you will find all the *gif* images from the *samples* directory.
-  **Note:** You can drag a resource from the **Open/Find Resource** view and drop it in a DocBook, DITA, TEI or XHTML document to create a link to that resource.

The **Open/Find Resource** view offers the following options:

- **Settings** - Displays settings for the view:
 - **Clear Index** - Clears the index.
 - **Show description** - Presents the search results in a more compact form, displaying only the title and the location of the resources.
 - **Options** - Opens the [Open/Find Resource preferences page](#).
- **In file paths** - Select this option to search for resources by their name or by its path (or a fragment of its path).
- **In content** - Select this option to search through the content of your resources.
- **In reviews** - Select this option to search through the comments, insertions, and deletion in your resources.
- **Reindex** - Reindexes your resources.

The Open/Find Resource Dialog Box

To open the **Open/Find Resource** dialog box, go to **Find >  Open/Find Resource ... Ctrl Shift R (Command Shift R on OS X)**. You can also click the  **Open/Find Resource ...** toolbar button or use the  **Search for file** action, available for some URL input fields.

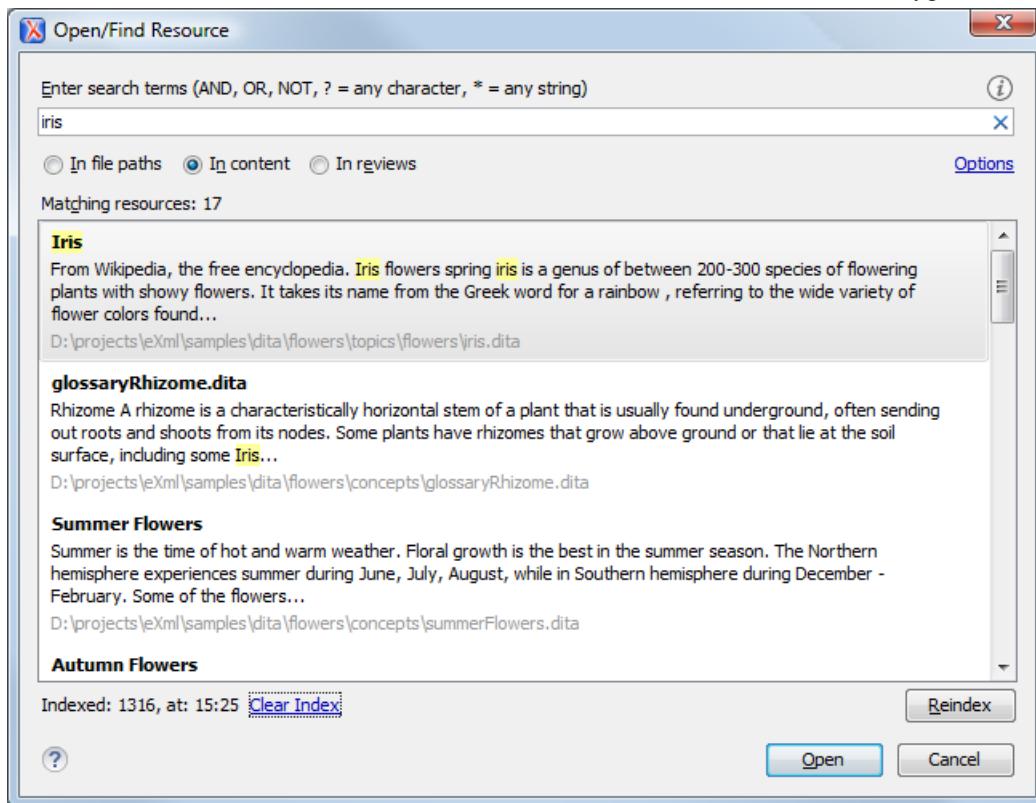


Figure 72: The Open/Find Resource Dialog Box

You can use this dialog box to find a file in the current Oxygen XML Editor project or in one of the DITA maps opened in [the DITA Maps Manager view](#) by typing a few letters of the file name of a document or a fragment of the content you are searching for. The **Open/Find Resource** dialog box also supports searching in document edits (comments, insertions, deletions, and highlighted content).

 **Note:** Full support for searching in document edits is available only in the Enterprise edition of Oxygen XML Author and Oxygen XML Editor. The Professional edition offers support to search through a maximum of 10 edits.

Search results are presented instantly, after you finish typing the content. The matching fragments of text are highlighted in the results list displayed in the dialog box. When you open one of the documents from the results list, the matching fragments of text are highlighted in the editing area. To remove the highlighting from your document close the [Results view](#). To display the search history, position the caret in the search field and press [Ctrl Down Arrow \(Command Down Arrow on OS X\)](#) or [Ctrl Up Arrow \(Command Up Arrow on OS X\)](#) on your keyboard. Pressing only the [Down Arrow](#) key moves the selection to the list of results.

 **Note:** Searches are case insensitive. For example, if you search for *car* you get the same results as when you search for *Car*.

 **Note:** Suffix searches are supported, both for searching in the content of your resources and in their name. For this, you can use wildcards. If you search **in content** for **ing* you will find documents that contain the word *presenting*. If you search **in file paths** for **/samples/*.gif* you will find all the gif images from the samples directory.

The **Open/Find Resource** dialog box offers the following options:

- **In file paths** - Select this option to search for resources by their name or by its path (or a fragment of its path).
- **In content** - Select this option to search through the content of your resources.
- **In reviews** - Select this option to search through the comments, insertions, and deletion in your resources.

- **Options** - Opens the [Open/Find Resource preferences page](#).
- **Clear Index** - Clears the index.
- **Reindex** - Reindexes your resources.

A contextual menu available on each search result provides actions applicable to the document that contains it. These actions allow you to:

- Open the document in one of Oxygen XML Editor internal editors.
- Open the document in an external system application.
- Identify the document in the system file explorer.
- Copy the file's location.

When you perform a search a caching mechanism is used to gather the paths of all files linked in the current project. When the first search is performed, all project files are indexed and added to the cache. The next search operations use the information extracted from the cache, thus improving the processing time. The cache is kept for the currently loaded project only, so when you perform a search in a new project the cache is rewritten. Also, the cache is reset when you press the **Reindex** button.

 **Important:** Files larger than 2GB are not indexed.

If there is no file found that matches your file pattern or text search, a possible cause is that the file you are searching for was added to the Oxygen XML Editor project after the last caching operation. In this case, re-indexing the project files from the **Reindex** button enables the file to be found. The date and time of the last index operation are displayed below the file list.

Once you find the files that you want to open, select them in the list and press the **Open** button. Each of the selected files is opened in [the editor associated with the type of the file](#).

To watch our video demonstration about the **Open/Find Resource** dialog box and search capabilities, go to http://oxygenvml.com/demo/Open_Find_Resource.html.

Searching in Content

To perform a search through the content of your resources, open the **Open/Find Resources** dialog or the **Open/Find Resource** view, enable the **in content** option, and in the search field enter the terms that you want to search for.

The **Open/Find Resource** feature is powered by [Apache Lucene](#). Apache Lucene is a free open source information retrieval software library.

You can use the **Open/Find Resource** dialog and the **Open/Find Resource** view either to perform a simple text search or to perform a more complex search using the [Apache Lucene - Query Parser Syntax](#). Using the [Apache Lucene - Query Parser Syntax](#) means you can perform any of the following searches:

Term Searches

Use the **Open/Find Resource** view or dialog to search for plain text:

```
Garden Preparation
```

Element Specific Searches

Use the **Open/Find Resource** view or dialog to search for content that belongs to a specific element:

```
title: "Garden Preparation"
```

Wildcard Searches

Use wildcards to make your search more permissive:

```
Garden Prepar?tion
```

Fuzzy Searches

In case you are not sure what is the exact form of a term that you are interested in, use the fuzzy search to find the terms that are similar to what you introduce in the **Open/Find Resource** view or dialog. To perform a fuzzy search, use the ~ symbol after the word that you are not sure of:

```
Garden Preparing~
```

Proximity Searches

Use proximity searches to find words that are within a specific distance away. To perform a proximity search, use the ~ symbol at the end of your search. For example, to search for the word *Garden* and the word *Preparation* within 6 words of each other use:

```
"Garden Preparation"~6
```

Range Searches

Use range searches to match documents whose element values are between the lower and upper bound specified in the range query. For example, to find all documents whose titles are between *Iris* and *Lilac*, use:

```
title:{Iris TO Lilac}
```

The curly brackets denote an exclusive query. The results you get when using this query are all the documents whose titles are between *Iris* and *Lilac*, but not including *Iris* and *Lilac*. To create an inclusive query use square brackets:

```
title:[Iris to Lilac]
```

Term Prioritising Searches

Use term prioritising searches in case the fragment of text that you are searching for contains certain words that are more important to your search than the rest of them. For example, if you are searching for *Autumn Flowers*, a good idea is to prioritise the word *Autumn* since the word *Flowers* occurs more often. To prioritise a word use the ^ symbol:

```
Autumn^6 Flowers
```

Searches Using Boolean Operators

You are able to use the **AND**, +, **OR**, -, and **NOT** operators.

To search for documents that contain both the words *Garden* and *Preparation*, use:

```
Garden AND Preparation
```

To search for documents that must contain the word *Garden* and may contain the word *Preparation*, use:

```
+Garden Preparation
```

To search for documents that contain either the word *Garden* or the word *Preparation*, use:

```
Garden OR Preparation
```

To search for documents that contain *Garden Preparation* but not *Preparation of the Flowers*, use:

```
"Garden Preparation" - "Preparation of the Flowers"
```

Searches Using Grouping

To search either for the word *Garden* or *Preparation*, and the word *Flowers*, use:

```
(Garden OR Preparation) AND Flowers
```

Searches Using Element Grouping

To search for a title that contains both the word *Flowers* and the phrase *Garden Preparation*, use:

```
title:(+Flowers +"Garden Preparation")
```

Searching in File Paths

To perform a search in the file paths of your resources, open the **Open/Find Resources** dialog or the **Open/Find Resource** view, enable the **In file paths** option, and in the search field enter the terms that you want to search for.

The **Open/Find Resource** view and dialog allows you to search for a resource either by its name or by its path (or by a fragment of its path).

You can use wildcards when you perform such searches:

- Use "*" to match any sequence of characters.
- Use "?" to match any single character.

For example, if you search for **-preferences-page* you will find all the resources that contain the *-preferences-page* fragment in their name. If you search for **/samples/*.gif*, you will find all the *.gif* images from the *samples* directory.

Searching in Reviews

To perform a search in the edits of your resources, open the **Open/Find Resource** dialog or the **Open/Find Resource** view, enable the **In reviews** option, and in the search field enter the terms that you want to search for.

The following options are available:

- **Type** - specifies whether you want to search for content in comments, insertions, deletions, or highlighted content;
- **Author** - displays all the authors of the edits in your resources. The authors are collected when indexing. You can set a specific author for your search or all of them;
- **Time** - specifies the time when the edits that you are searching through were created.

Both the view and the dialog display the edits that contain the search results and their parent topics along with a short description. To hide this description, go to **Settings** and disable the **Show Description** option.

Technical Aspects

When Oxygen XML Editor performs the indexing of your resources, the refereed content from your documents is not taken into account. For example, when DITA documents are indexed, the content from the `conref` elements is not

parsed. The files that make up the index are stored on the disk in the
 C:\Users\USER_NAME\AppData\Roaming\com.oxygenxml\lucene folder.

Opening Local Files at Start-up from Command Line

To open a local file at start-up when you open Oxygen XML Editor from the command line, add the paths for one or more local files as parameters in the command line:

- scriptName [pathToXMLFile1] [pathToXMLFile2] ... where scriptName is the name of the startup script for your platform (oxygen.bat on Windows, oxygen.sh on Unix/Linux, oxygenMac.sh on Mac OS) and pathToXMLFileN is the name of a local XML file.
- an XML file and a schema file to be associated automatically to the file and used for validation and content completion:

```
scriptName -instance pathToXMLFile -schema pathToSchemaFile -schemaType
XML_SCHEMA|DTD_SCHEMA|RNG_SCHEMA|RNC_SCHEMA -dtName documentTypeName
```

where scriptName is the name of the startup script for your platform (oxygen.bat on Windows, oxygen.sh on Unix/Linux, oxygenMac.sh on Mac OS), pathToXMLFile is the name of a local XML file, pathToSchemaFile is the name of the schema which you want to associate to the XML file, the four constants (XML_SCHEMA, DTD_SCHEMA, RNG_SCHEMA, RNC_SCHEMA) are the possible schema types (W3C XML Schema, DTD, Relax NG schema in full syntax, Relax NG schema in compact syntax). The next parameter, documentTypeName, specifies the name of the *Document Type* for which the schema is defined. If the Document Type is already set in preferences, its schema and type are updated.

The two possibilities of opening files at startup by specifying them in the command line are explained also if the startup script receives one of the *-h* or *--help* parameters.

Opening a File at a Specific Position Using the Command Line Interface

Oxygen XML Editor offers support for opening a file at a specific position using the command line interface, by transmitting parameters to the Oxygen XML Editor batch script file. The following methods are available, depending on how you identify the position needed:

1. Specific position values (line and column number , or character offset)

Oxygen XML Editor supports the following position parameters:

- line - The line number
- column - The column number (has meaning if the line parameter is also defined)
- char - The character offset

Examples for Windows:

The following examples show how you can open an XML document in Oxygen XML Editor:

```
oxygen.bat file:samples/personal.xml#line=4
oxygen.bat file:samples/personal.xml#line=4column=5
oxygen.bat file:samples/personal.xml#line=4;column=5
oxygen.bat file:samples/personal.xml#char=334
```

2. Simplified XPath index path

Oxygen XML Editor will open an XML file and select one of its elements identified by a simplified XPath index path. For example, an index path of the form 1/5/7 identifies the seventh child of the fifth child of the root element.

Examples for Windows:

The following example shows how you can open an XML document in Oxygen XML Editor and select the third child of the root element:

```
oxygen.bat file:samples/personal.xml#element(1/3)
```

3. Anchors identified by ID attribute values

Oxygen XML Editor will open an XML file and select the element whose `id` attribute value is an exact match of the anchor attached to a command line instruction.

Examples for Windows:

The following example shows how you can open an XML document in Oxygen XML Editor and select the element that has the `id` element set to `titleID`:

```
oxygen.bat  file:samples/personal.xml#titleID
```

Opening and Saving Remote Documents via FTP/SFTP/WebDAV/SharePoint

Oxygen XML Editor supports editing remote files, using the FTP, SFTP ,WebDAV, SharePoint, and SharePoint Online for Office 365 protocols. You can edit remote files in the same way you edit local files. For example, you are able to add remote files a project, or make them subject of XSL and FO transformations.

You can open one or more remote files in [the Open URL dialog box](#).

A WebDAV resource can be locked when it is opened in Oxygen XML Editor by checking the [**Lock WebDAV files on open**](#) option to prevent other users to modify it concurrently on the server. If a user tries to edit a locked file, Oxygen XML Editor displays an error message that contains the lock owner's name. The lock is released automatically when the editor for that resource is closed in Oxygen XML Editor.

To avoid conflicts with other users when you edit a resource stored on a SharePoint server, you can **Check Out** the resource.

To improve the transfer speed, the content exchanged between Oxygen XML Editor and the HTTP / WebDAV server is compressed using the GZIP algorithm.

The current [WebDAV Connection](#) details can be saved using the  **Database Perspective** button and then used in the *Data Source Explorer* view.

The Open URL Dialog

To open this dialog, go to **File >  Open URL ...** (or click the  **Open URL ...** toolbar button), then choose the  **Browse for remote file** option from the drop down action list.

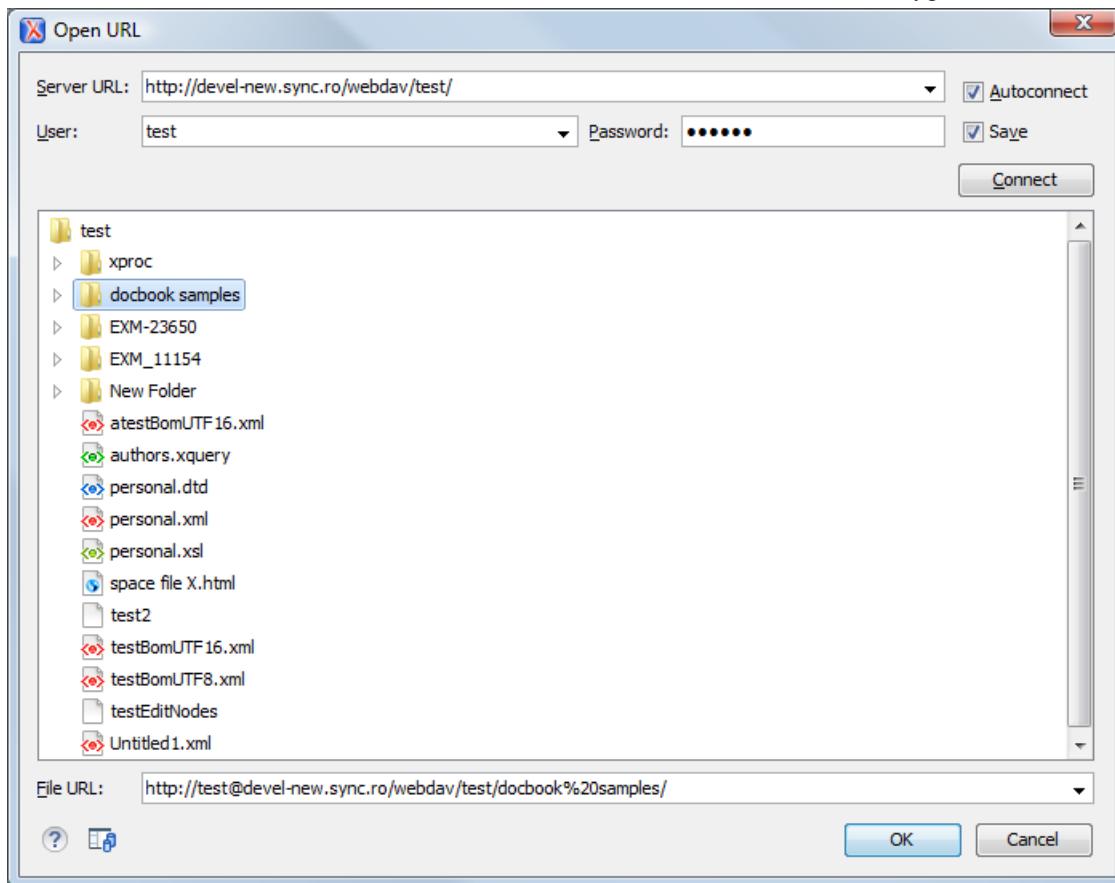


Figure 73: Open URL Dialog

The displayed dialog is composed of several parts:

- The *Identification* section contains the access credentials. To browse for a file on a server, you have to specify the user and password. This information is bound to the selected URL displayed in the **File URL** combo box, and used further in opening/saving the file. If the check box **Save** is selected, then the user and password are saved between editing sessions. The password is kept encrypted in the options file.



Note:

Your password is well protected. In the case the options file is used on other machine by a user with a different username the password will become unreadable, since the encryption is username dependent. This is also true if you add URLs having user and password to your project.

- In the server combo you can specify the protocol (HTTP, HTTPS or FTP) and the host name or IP of the server.



Tip: When specifying a URL, follow these rules:

- to access an FTP server, write the protocol, host, and port (if using a non-standard one), like
ftp://server.com or ftp://server.com:7800/
- to access a WebDAV server, write the path to the directory of the WebDAV repository along with the protocol and the host name, like
https://www.some-webdav-server.com:443/webdav-repository/



Important:

Make sure that the repository directory ends in a slash "/", like

https://www.some-webdav-server.com:443/webdav-repository/

By pressing the **Connect** button the directory listing will be shown in the component below. If the input URL points to a SharePoint server, a dedicated SharePoint browsing component is presented. When **Autoconnect** is selected, the browse action is performed every time when you open the dialog.

- The browser view:
 - In case you are browsing a WebDAV or FTP repository, the items are presented in a tree-like fashion. You can browse the directories, and make multiple selections. Additionally, you may use the **Rename**, **Delete**, and **New Folder** actions to manage the file repository.
- Note: The file names are sorted in a case-insensitive way.
- When you browse a SharePoint repository, a specialized component renders the SharePoint site content.

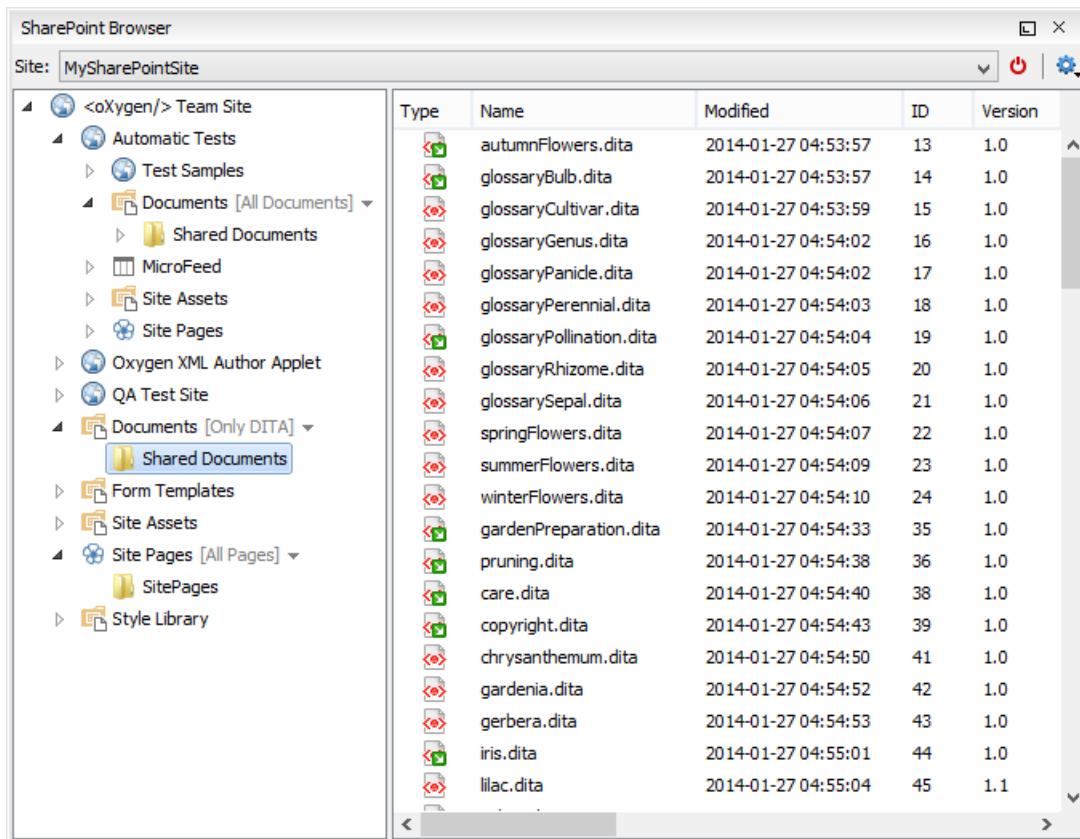


Figure 74: Browsing a SharePoint Repository

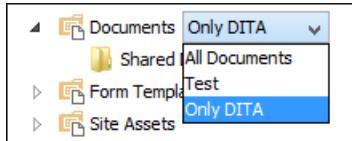
The left side navigation area presents the SharePoint site structure in a tree-like fashion displaying the following node types: *sites* (🌐), *libraries*, and *folders*.

Depending on a node's type, a contextual menu offers customized actions that can be performed on that node.

Note: The contextual menu of a folder allows you to create new folders, new documents, and to rename and delete the folder.

Note: The rename and delete actions are not available for library root folders (folders located at first level in a SharePoint library).

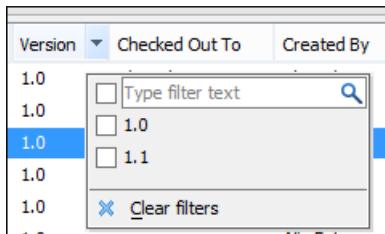
Each library node display next to its name a drop down box where you can select the current library *view*. This functionality is also available on the node's contextual menu, under the **Current View** submenu.



The content of a folder is displayed in a tabular form, where each row represents the properties of a folder or document. The list of columns and the way the documents and folders are organized depends on the currently selected view of the parent library.

You can filter and sort the displayed items. To display the available filters of a column, click the filter widget located on the column's header. You can apply multiple filters at the same time.

Note: A column can be filtered or sorted only if it was configured this way on the server side.



- The editable combo box, in which it can be specified directly the URL to be opened or saved.



Tip:

You can type in here an URL like `http://some.site/test.xml`, in case the file is accessible through normal HTTP protocol, or `ftp://anonymous@some.site/home/test.xml` if the file is accessible through anonymous FTP.

This combo box is also displaying the current selection when the user changes selection by browsing the tree of folders and files on the server.

Changing File Permissions on a Remote FTP Server

Some FTP servers allow the modification of permissions of the files served over the FTP protocol. This protocol feature is accessible directly in the FTP/WebDAV file browser dialog box by right-clicking on a tree node and selecting the *Change permissions* menu item.

In this dialog box, the usual Unix file permissions *Read*, *Write*, and *Execute* are granted or denied for the file owner, owner group, and the rest of the users. The aggregate number of permissions is updated in the *Permissions* text field when it is modified with one of the check boxes.

WebDAV over HTTPS

If you want to access a WebDAV repository across an insecure network, Oxygen XML Editor allows you to load and save the documents over the HTTPS protocol (if the server understands this protocol) so that any data exchange with the WebDAV server is encrypted.

When a WebDAV repository is first accessed over HTTPS, the server hosting the repository will present a security certificate as part of the HTTPS protocol, without any user intervention. Oxygen XML Editor will use this certificate to decrypt any data stream received from the server. For the authentication to succeed you should make sure the security certificate of the server hosting the repository can be read by Oxygen XML Editor. This means that Oxygen XML Editor can find the certificate in the key store of the Java Runtime Environment in which it runs. You know the server certificate is not in the JRE key store if you get the error *No trusted certificate found* when trying to access the WebDAV repository.

Troubleshooting HTTPS

When Oxygen XML Editor cannot connect to an HTTPS-capable server, most likely there is no certificate set in the *Java Runtime Environment (JRE)* that Oxygen XML Editor runs into. The following procedure describes how to:

- export a certificate to a local file using any HTTPS-capable Web browser (for example Internet Explorer)
- import the certificate file into the JRE using the keytool tool that comes bundled with Oxygen XML Editor

1. Export the certificate into a local file

- a) Point your HTTPS-aware Web browser to the repository URL.

If this is your first visit to the repository it will be displayed a security alert stating that the security certificate presented by the server is not trusted.

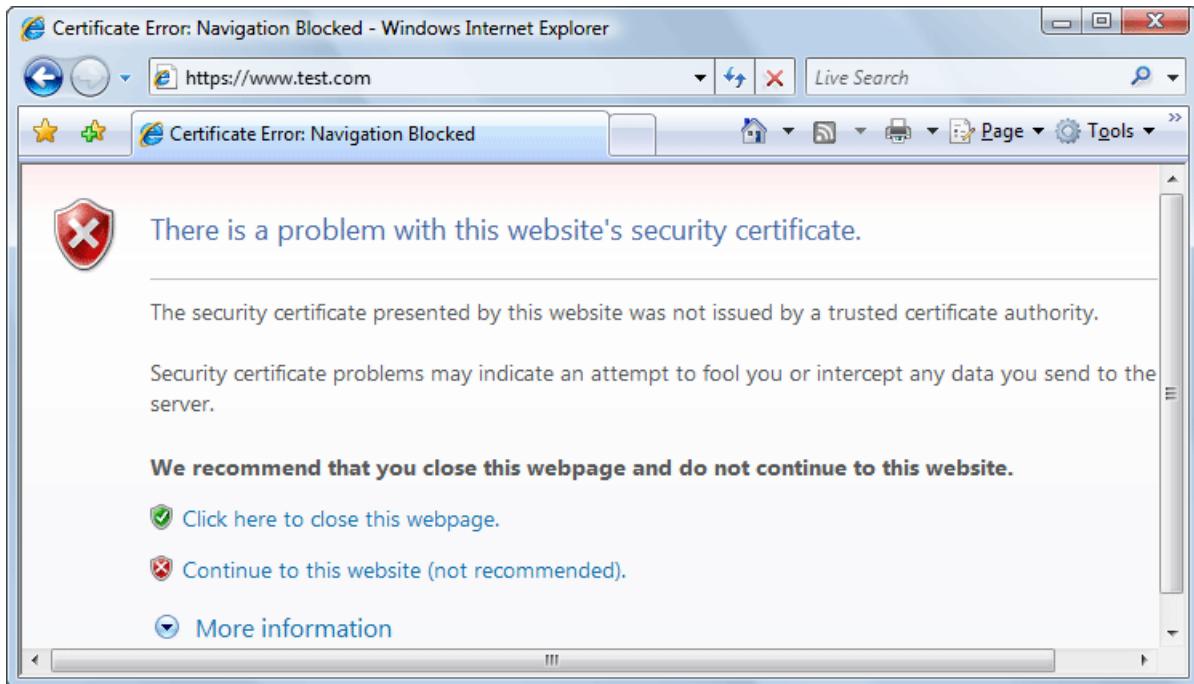


Figure 75: Security alert - untrusted certificate

- b) Go to menu **Tools > Internet Options**.
Internet Options dialog box is opened.
 - c) Select **Security** tab.
 - d) Select **Trusted sites** icon.
 - e) Press **Sites** button.
This will open **Trusted sites** dialog box.
 - f) Add repository URL to **Websites** list.
 - g) Close the **Trusted sites** and **Internet Options** dialog boxes.
 - h) Try again to connect to the same repository URL in Internet Explorer.
The same error page as above will be displayed.
 - i) Select **Continue to this website** option.
A clickable area with a red icon and text **Certificate Error** is added to Internet Explorer address bar.
 - j) Click on **Certificate Error** area.
A dialog box containing a **View certificates** link is displayed.
 - k) Click on **View certificates** link.
Certificate dialog box is displayed.
 - l) Select **Details** tab of **Certificate** dialog box.
 - m) Press **Copy to File** button.
Certificate Export Wizard is started.
 - n) Follow indications of wizard for DER encoded binary X.509 certificate. Save certificate to local file **server.cer**.
- 2. Import the local file into the JRE running Oxygen XML Editor.**
- a) Open a text-mode console with administrative rights.

- b) Go to the lib/security directory of the JRE running Oxygen XML Editor. You find the home directory of the JRE in the `java.home` property that is displayed in the **About** dialog box (**System properties** tab). On Mac OS X systems, the lib/security directory is usually located in `/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home` directory.
- c) Run the following command:

```
...\\bin\\keytool -import -trustcacerts -file server.cer -keystore cacerts
```

The `server.cer` file contains the server certificate, created during the previous step. `keytool` requires a password before adding the certificate to the JRE `keystore`. The default password is `changeit`. If somebody changed the default password then he is the only one who can perform the import.

 **Note:** To make Oxygen XML Editor accept a certificate even if it is invalid, [open the Preferences dialog box](#), go to **Connection settings > HTTP(S)/WebDAV**, and enable the **Automatically accept a security certificate, even if invalid** option.

 **Tip:** If you need to import multiple certificates, you need to specify a different alias for each additional imported certificate with the `-alias` command line argument, like in the following example:

```
...\\bin\\keytool -import -alias myalias1 -trustcacerts -file server1.cer -keystore cacerts
...\\bin\\keytool -import -alias myalias2 -trustcacerts -file server2.cer -keystore cacerts
```

3. Restart Oxygen XML Editor.

Single Sign-on

Oxygen XML Editor implements the *Single sign-on* property, meaning that you can log in once and gain access to multiple services without being prompted to log in for each of them. The implementation is based on the *Kerberos* protocol and relies on a *ticket-granting ticket (TGT)* that Oxygen XML Editor obtains from the operating system.

To turn on the *Kerberos*-based authentication, you need to add the following system property in the `.vmoptions` configuration file:

```
-Djavax.security.auth.useSubjectCredsOnly=false
```

Opening the Current Document in System Application

To open the currently edited document in the associated system application, use the  **View in Browser/System Application** action that is available in the **File** menu and on the **File** toolbar. If you want to open XML files in a specific internet browser, instead of the associated system application, you can specify the internet browser to be used. To do so, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field. This will take precedence over the default system application settings.

Switching Between Opened Tabs

There are two actions for cycling through the opened file tabs:

Ctrl Tab (Command Tab on OS X)

Switches between the tabs with opened files in the order most recent ones first.

Ctrl Shift Tab (Command Shift Tab on OS X)

Switches between the tabs with opened files in the reverse order.

Closing Documents

To close open documents, use one of the following methods:

- Go to menu **File > Close (Ctrl W (Command W on OS X))**: Closes only the selected tab. All other tab instances remain opened.

- Go to menu **File > Close All (Ctrl Shift F4 (Command Shift F4 on OS X))**: If you try to close a modified or a newly created document, you are first prompted to save it.
- Click **Close** in the contextual menu of an open tab to close it.
- Click **Close Other Files** in the contextual menu of an open tab to close all the open tabs except the selected one.
- Click **Close All** in the contextual menu of an open tab to close all open tabs.

The Contextual Menu of the Editor Tab

The contextual menu is available when clicking the current editor tab label. It shows the following actions:

Close

Closes the current editor.

Close Other Files

Closes all opened editor but the one you are currently viewing.

Close All

Closes all opened editors.

Reopen last closed editor

Reopens the last closed editor.

Maximize/Restore Editor Area

Collapses all the side views and spans the editing area to cover the entire width of the main window.

Add to project

Adds the file you are editing to the current project.

Add all to project

Adds all the opened files to the current project.

Copy Location

Copies the disk location of the file.

Show in Explorer (Show in Finder on OS X)

Opens the Explorer to the file path of the file.

Viewing File Properties

In the **Properties** view, you can quickly access information about the current edited document like:

- character encoding
- full path on the file system
- schema used for content completion and document validation
- document type name and path
- associated transformation scenario
- file's read-only state
- bidirectional text (left to right and right to left) state
- total number of characters in the document
- line width
- indent with tabs state
- indent size

The view can be accessed from **Window > Show View > Properties**.

To copy a value from the **Properties** view in the clipboard, for example the full file path, use the **Copy** action available on the contextual menu of the view.

Grouping Documents in XML Projects

This section explains how to create and work with projects.

Using the Project View

The **Project** view is designed to assist the user in organizing and managing related files grouped in the same XML project. The actions available on the context menu and toolbar associated to this panel, enable the creation of XML projects and shortcuts to various operations on the project documents.

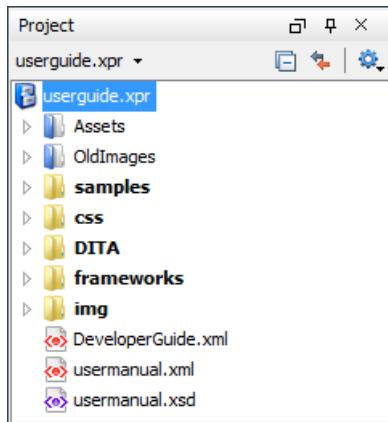


Figure 76: The Project View

By default, the **Project** view is positioned on the left side of the Oxygen XML Editor window, above the **Outline view**. A closed view can be quickly reopened at any time with the **Project > Show Project View** menu action.

The tree structure occupies most of the view area. In the upper left side of the view, there is a drop-down list that contains all recently used projects and project management actions:

Open Project ... (Ctrl F2 (Command F2 on OS X))

Opens an existing project. Alternatively, you can open a project by dropping an Oxygen XML Editor XPR project file from the file explorer into the **Project** panel.

New Project...

Creates a new, empty project.

The following actions are grouped in the upper right corner:

Collapse All

Collapses all project tree folders. You can also collapse/expand a project tree folder if you select it and press the **Enter** key or **Left Arrow** to collapse and **Right Arrow** to expand.

Link with Editor

When selected, the project tree highlights the currently edited file, if it is found in the project files.

Note: This button is disabled automatically when you move to the **Debugger** perspective.

Settings

A sub-menu that contains the following actions:

Filters...

Allows you to filter the information displayed in the **Project** view. Click the toolbar button to set filter patterns for the files you want to show or hide. Also, you can set filter patterns for the linked directories that are hidden.

Show Full Path

When selected, linked files and folders are presented with a full file path.

Enable Master Files Support

Select this option to enable the *Master Files support*.

Change Search and Refactor operations scope...

Allows you to change the collection of documents that define the context of the *search and refactor operations*.

- **Use only Master Files, if enabled** - Restricts Oxygen XML Editor to perform the search and refactor operations starting from the master files that are defined for the current resource. This option is available when you select **Project** in the **Select the scope for Search and Refactor operations** dialog and the **Master Files** support is enabled.
- **Working sets** - Allow you to specify the set of files on which the search and refactor operations will act on.

The files are usually organized in an XML project as a collection of folders. There are three types of resources displayed in the **Project** view:

- *Logical folders* - marked with a blue icon on Windows and Unix/Linux (📁) and a magenta icon on Mac OS X (📁). They help you group files within the project. This folder type has no correspondent on the physical disk, since they are used as containers for related items. Creating and deleting them does not affect the file system on disk. They are created on the project root or inside other logical folders by using the contextual action **New > Logical Folder...**. The contextual menu action ✖ **Remove from Project** can be used to remove them from the project.
- *Physical folders and files* - marked with the operating system-specific icon for folders (usually a yellow icon on Windows and a blue icon on Mac OS X). These folders and files are mirrors of real folders or files that exist in the local file system. They are created or added to the project by using contextual menu actions (such as **New > File**, **New > Folder**, **Add Folder...**, etc.) Also, the contextual menu action 🗑 **Remove from Disk (Shift+Delete)** can be used to remove them from the project and local file system.
- *Shortcut folders and files* - the icons for file shortcuts include a shortcut symbol and names of folder shortcuts are displayed in bold text. All files and folders that appear as direct descendants of a logical folder are considered shortcuts. They are created and added with the contextual actions **Add Files...** and **Add Folder...** from the project root. Both contextual menu actions ✖ **Remove from Project** and 🗑 **Remove from Disk (Shift+Delete)** are available for shortcuts. ✖ **Remove from Project** just removes the shortcut from the project, while 🗑 **Remove from Disk (Shift+Delete)** removes the shortcut and the physical resource from the local file system.

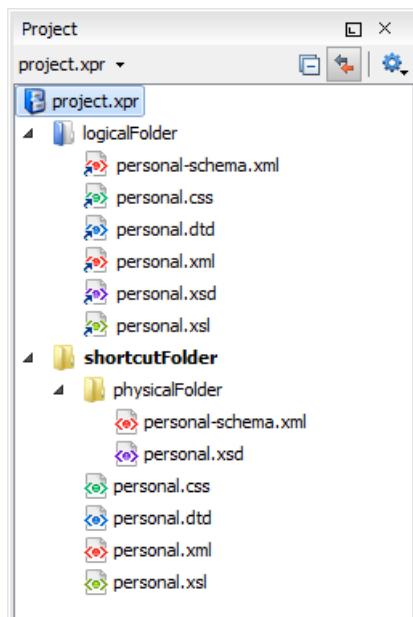


Figure 77: The Project View with Examples of all Three Types of Resources

Creating New Project Items

The following actions are available by selecting **New** from the contextual menu, when invoked from the project root:

New > File...

Creates a new file and adds it to the project structure.

New > Logical Folder...

Creates a logical folder in the tree structure (the icon is a magenta folder on Mac OS X - ).

New > Logical Folders from Web...

Replicates the structure of a remote folder accessible over FTP/SFTP/WebDAV, as a structure of logical folders.

The newly created logical folders contain the file structure of the folder it points to.

New Project...

Creates a new, empty project.

Add Content to a Logical Folder

The project itself is considered a logical folder. You can add content to a logical folder using one of the actions available in the contextual menu:

Add Folder...

Adds a link to a physical folder, whose name and content mirror a real folder that exists in the local file system (the icon of this action is different on Mac OS X ).

Add Files...

Adds links to files on the local file system.

Add Edited File

Adds a link to the currently edited file in the project.

Managing Project Content

Creating/Adding Files and Folders

You can create linked folders (shortcuts) by dragging and dropping folders from the Windows Explorer / Mac OS X Finder to the project tree, or by selecting **Add Folder...** in the contextual menu from the project root. To create a file inside a linked folder, select the **New > File...** action from the contextual menu.

 **Note:** The linked files presented in the **Project** view are marked with a special icon. Linked folders are displayed in bold text.

You can create physical folders by selecting **New > Folder...** from the contextual menu.

When adding files to a project, the default target is the project root. To change a target, select a new folder. Files may have multiple instances within the folder system, but cannot appear twice within the same folder.

Removing Files and Folders

To remove one or more linked files or folders, select them in the project tree and press the **Delete** key, or select the contextual menu action  **Remove from Project**. To remove a linked file or folder from both project and local file system, select the contextual menu action  **Remove from Disk (Shift+Delete)**. The  **Remove from Disk (Shift+Delete)** action is also used to remove physical files or folders.

 **Caution:** In most cases this action is irreversible, deleting the file permanently. Under particular circumstances (if you are running a Windows installation of Oxygen XML Editor and the *Recycle Bin* is active) the file is moved to *Recycle Bin*.

Moving Files and Folders

You can *move the resources of the project* with drag and drop operations on the files and folders of the tree.

You can also use the usual  **Cut**,  **Copy**, and  **Paste** actions to move resources in the **Project** view.

Renaming Files and Folders

There are three ways you can *rename an item in the Project view*. Select the item in the **Project** view and do one of the following:

- Invoke the **Rename** action from the contextual menu.
- Press **F2** and type the new name.
- Click the selected item and type the new name.

To finish editing the item name, press **Enter**.

 **Note:** The **Rename** action is also available on *logical* files.

Locating and Opening Files

If a project folder contains a lot of documents, a certain document can be located quickly in the project tree by selecting the folder containing the desired document and typing the first few characters of the document name. The desired document is automatically selected as soon as the typed characters uniquely identify its name in the folder.

The selected document can be opened by pressing the **Enter** key, by double-clicking it, or with one of the **Open** actions from the contextual menu. The files with known document types are opened in the associated editor, while binary files are opened with the associated system application. To open a file with a known document type in an editor other than the default one, use the **Open with** action. Also, dragging and dropping files from the project tree to the editor area results in the files being opened.

Saving the Project

The project file is automatically saved every time the content of the **Project** view is saved or modified by actions such as adding or removing files and drag and drop.

Validate Files

The currently selected files in the **Project** view can be checked to be XML well-formed or validated against a schema (DTD, XML Schema, Relax NG, Schematron or NVDL) with one of the following contextual menu actions found in the **Validate** sub-menu:

Check Well-Formedness

Checks if the selected file or files are well-formed.

Validate

Validates the selected file or files against their associated schema. EPUB files make an exception, because this action triggers a *Validate and Check for Completeness* operation.

Validate with Schema...

Validates the selected file or files against a specified schema.

Configure Validation Scenario(s)...

Allows you to configure and run a *validation scenario*.

Applying Transformation Scenarios

The currently selected files in the **Project** view can be transformed in one step with one of the following actions available from contextual menu in the **Transform** sub-menu:

Transform > Apply Transformation Scenario(s)

Obtains the output with one of the built-in scenarios.

Transform >  Configure Transformation Scenario(s)...

Opens a dialog that allows you to configure pre-defined transformation scenarios.

Transform >  Transform with...

Allows you to select a transformation scenario to be applied to the currently selected files.

Along with the logical folder support, this allows you to group your files and transform them very easily.

Other Contextual Menu Actions

Other actions that are available in the contextual menu from the project tree include:

 Open

Displays the **Open** file dialog.

Open with submenu

This submenu allows you to open the selected file with the internal editor, a system application, or other internal tools: *DITA Maps Manager*, *Archive Browser*, *MathML Editor*, *Generate/Convert Schema*, *WSDL/SOAP Analyzer*, *Large File Viewer*, *Hex Viewer*, *SVG Viewer*.

Open All Files (when a folder or multiple files/folders are selected)

Opens all the selected files with the corresponding editors.

Show in Explorer (or Show in Finder on OS X)

In Windows, the content of the selected folder or file is presented in a specific explorer window. On MAC OS X, the parent folder is opened and the selected folder is highlighted in a specific finder window.

Refactoring > Rename resource... (Available for certain document types (such as XML, XSD, and XSL files))

Allows you to change the name of a resource.

Refactoring > Move resource... (Available for certain document types (such as XML, XSD, and XSL files))

Allows you to change the location on disk of a resource.

Refactoring >  XML Refactoring...

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

 Refresh

Refreshes the content and the dependencies between the resources in *the Master Files directory*.

 Find/Replace in Files...

Allows you to *find and replace text in multiple files*.

 XPath in Files...

Opens the *XPath/XQuery Builder* view that allows you to compose XPath and XQuery expressions and execute them over the currently edited XML document.

 Open/Find Resource...

Opens the *Open/Find Resource dialog box*.

 Check Spelling in Files...

Allows you to *check the spelling of multiple files*.

 Format and Indent

Opens the **Format and Indent** dialog box that allows you to configure the format and indent (pretty print) action that will be applied on the selected document.

 Open in SVN Client...

Syncro SVN Client tool is opened and it highlights the selected resource in its corresponding working copy.

Compare...

Opens [the Compare Directories or Compare Files tool](#).

Generate Documentation > XML Schema Documentation...

Opens [the XML Schema Documentation Dialog Box](#).

Generate Documentation > XSLT Stylesheet Documentation...

Opens [the XSLT Stylesheet Documentation Dialog Box](#).

Generate Documentation > XQuery Documentation...

Opens [the XQuery Documentation Dialog Box](#).

Generate Documentation > WSDL Documentation...

Opens [the WSDL Documentation Dialog Box](#).

Properties

Displays the properties of the current file in a **Properties** dialog box.

Menu Level Actions

The following actions are available in the **Project** menu:

New Project...

Creates a new, empty project.

Open Project ... (**Ctrl F2 (Command F2 on OS X)**)

Opens an existing project. Alternatively, you can open a project by dropping an Oxygen XML Editor XPR project file from the file explorer into the **Project** panel.

Save Project As...

Allows you to save the current project under a different name.

Validate all project files

Checks if the project files are well-formed and their mark-up conforms with the specified DTD, XML Schema, or Relax NG schema rules. It returns an error list in the message panel.

Filters

Opens the **Project filters** dialog that allows you to decide which files and directories will be shown or hidden.

Enable Master Files Support

Allows you to enable the [Master Files Support](#) for each project you are working on.

Change Search and Refactor operations scope

Opens a dialog that allows you to define the context of search and refactor operations.

Show Project View

Displays the project view.

Reopen Project

Contains a list of links of previously used projects. This list can be emptied by invoking the **Clear history** action.

Team Collaboration - Apache Subversion™

To assist you with team collaboration and sharing projects, Oxygen XML Editor includes an embedded [SVN \(Subversion Client\)](#). It can be accessed from the **Tools** menu and can be used for synchronizing your working copy with a central repository.

It can also be started by selecting the  **Open in SVN Client** action from the contextual menu of the **Project** view. This action opens the Syncro SVN Client and shows the selected project file in the **Working Copy** view.

Minimize Differences Between Versions Saved on Different Computers

The number of differences between versions of the same file saved by different content authors on different computers can be minimized by imposing the same set of formatting options when saving the file, for all the content authors. An example for a procedure that minimizes the differences is the following.

1. Create an Oxygen XML Editor project file that will be shared by all content authors.
2. Configure your own formatting preferences. To do this, [open the Preferences dialog box](#), go to **Editor > Format**, configure the appropriate options in this page, then go to **Editor > Format > XML** and configure the options there.
3. Save the preferences of these two panels in the Oxygen XML Editor project by selecting the button **Project Options** in these two panels.
4. Save the project and commit the project file to your versioning system so all the content authors can use it.
5. Make sure the project is opened in the **Project** view.
6. Open and save your XML files in the **Author** mode.
7. Commit the saved XML files to your versioning system.

When other content authors will change the files only the changed lines will be displayed in your diff tool instead of one big change that does not allow to see the changes between two versions of the file.

Project Level Settings

You can not only store files and directories into the project, but also transformation scenarios and other settings specific to that project. For more information, see the [Preference Sharing](#) and [Sharing the Transformation Scenarios](#) topics.

Moving/Renaming Resources in the Project View

The **Project** view allows you to move or rename a file from the current project.

To move a file or a directory, drag and drop it to the new location in the tree structure from the **Project** view. You can also use the usual  **Cut**,  **Copy**, and  **Paste** actions or right click the file or directory and select the **Refactoring > Move resource...** action from its contextual menu. Oxygen XML Editor presents a **Move resource** dialog box with the following fields available:

- **Destination** - presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - enable this option to update the references to the resource you are moving, in accordance with the new location and name.

To quickly rename a file or a directory, use the in-place editing either by pressing **F2** or by selecting **Rename** from the contextual menu of the resource. You can also right click a file or a directory and select the **Refactoring > Rename resource** action from its contextual menu. Oxygen XML Editor presents the **Rename** dialog box in case you used in-place editing. The **Rename resource** dialog box is presented if you used the refactoring actions. The following fields are available:

- **New name** - presents the current name of the edited resource and allows you to modify it.
- **Update references of the renamed resource** - enable this option to update the references to the resource you are renaming.
- **Scope** - specifies the [scope of the rename operation](#).



Note: The support to update references is available for XML, XML Schema, XSLT, Relax NG, and WSDL documents.

Problems with Updating References of Moved/Renamed Resources

In some cases the references of a moved or a renamed resource can not be updated. For example, when a resource is resolved through an XML catalog or when the path to the moved or renamed resource contains entities. For these cases, Oxygen XML Editor displays a warning dialog.

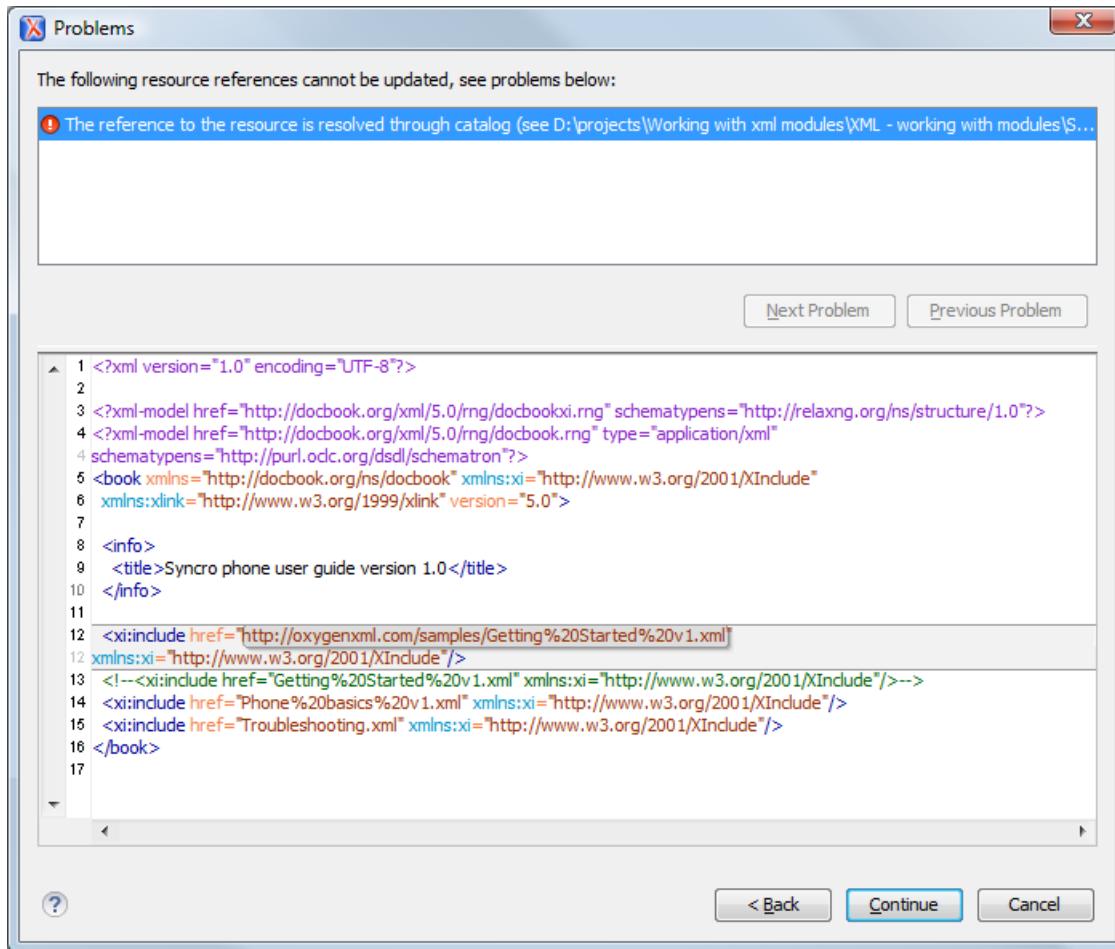


Figure 78: Problems Dialog

Defining Master Files at Project Level

This chapter details the **Master Files Support** available in Oxygen XML Editor.

The **Master Files Support** helps you simplify the configuration and development of XML projects. A *Master File* typically refers to the root of an import/include tree of modules.

Introduction

Oxygen XML Editor allows you to define *master files* at project level. These *master files* are automatically used by Oxygen XML Editor to determine the context for operations such as validation, content completion, refactoring, or search for XML, XSD, XSL, WSDL, and RNG modules. Oxygen XML Editor maintains the hierarchy of the *master files*, helping you to determine the editing context.

To watch our video demonstration about the **Master Files Support** for XML documents, XSL documents, and WSDL documents, go to [Working with Modular XML Files](#), [Master Files Support](#), and [Working with Modular WSDL Files](#), respectively.

Master Files Benefits

When you edit a module after defining the *master files*, you have the following benefits:

- When the module is validated, Oxygen XML Editor automatically identifies the *master files* that include the module and validates all of them.
- The **Content Completion Assistant** presents all the components that are collected, from the *master files* to the modules they include.

- The **Outline** view displays all the components that are defined in the *master files* hierarchy.
- The *master files* that are defined for the current module determines the *scope of the search and refactoring actions*. Oxygen XML Editor performs the search and refactoring actions in the context that the *master files* determine, thus improving the speed of execution.

Enabling the Master Files Support

Oxygen XML Editor stores the *master files* in a folder located in the **Project** view, as the first child of the project root. The **Master Files Support** is disabled by default. To enable the **Master Files Support**, go to the **Settings** menu of the **Project** view and select **Enable Master Files Support**. You can also select **Enable Master Files Support** from the contextual menu of the `Master Files` directory, or from the contextual menu of the project itself. Oxygen XML Editor allows you to enable/disable the **Master Files Support** for each project you are working on.

When the **Master Files Support** is disabled, Oxygen XML Editor displays a window tip located at the bottom of the project view. This window contains an **Enable** button and a **read more...** option. Clicking the **read more...** option takes you to the user guide. Clicking the **Enable** button opens the **Enable Master Files** dialog box. This dialog box contains general information about the **Master Files Support** and allows you to enable it. You can also use the **Detect and Enable** option to detect the *master files* from the current project.

 **Note:** Once you close this window tip, Oxygen XML Editor hides it for all projects. To make the window tip reappear, restore Oxygen XML Editor to its default settings.

 **Warning:** Restoring Oxygen XML Editor to its default settings causes loss of your customized options.

Detecting Master Files

Oxygen XML Editor allows you to detect the *master files* using the  **Detect Master Files...** option available in the contextual menu of the project. This action applies to the folders you select in the project. To detect *master files* over the entire project, do one of the following:

- Right-click the root of the project and select  **Detect Master Files...**.
- Use the  **Detect Master Files from Project...** option, available in the contextual menu of the `Master Files` folder.

Both of these options display the **Detect Master Files** wizard. In the first panel you can select the type of *master files* you want Oxygen XML Editor to detect. In the subsequent panel the detected *master files* are presented in a tree-like fashion. The resources are grouped into three categories:

- Possible *master files* - the files presented on the first level in this category are not imported/included from other files. These files are most likely to be set as *master files*.
- Cycles - the files that are presented on the first level have circular dependencies between them. Any of the files presented on the first level of a cycle is a possible *master file*.
- Standalone - files that do not include/import other files and are also not included/imported themselves. It is not necessary to set them as *master files*.

To set them as *master files*, enable their check-boxes. Oxygen XML Editor marks all the children of a *master file* as modules. Modules are rendered in gray and their tool-tip presents a list of their *master files*. A module can be accessed from more than one *master file*.

The *master files* that are already defined in the project are automatically marked in the tree and cannot be removed. The only way to disable a *master file* is to delete it from the `Master Files` folder.

The next panel displays a list with the selected *master files*. Click the **Finish** button to add the *master files* in the `Master Files` folder.

You can use the **Select Master Files** option to automatically mark all *master files*. This action sets all the resources from the **Possible Master Files** category and the first resource of each **Cycle** as *master files*.



Tip: We recommend you to only add top-level files (files that are at the root of the include/import graph) in the `Master Files` directory. Keep the file set to a minimum and only add files that import or include other files.



Attention: If the **Master Files Support** is disabled, the `Master Files` directory is rendered only if it contains *master files*.

Adding/Removing a Master File

The `Master Files` directory only contains logical folders and linked files. To add files in the `Master Files` directory, use one of the following methods:

- Right-click a file from your project and select **Add to Master Files** from the contextual menu.
- Select **Add Files** or **Add Edited File** from the contextual menu of the `Master Files` directory.
- Drag and drop files into the `Master Files` directory.
- From the contextual menu of the **Resource Hierarchy Dependencies** view, use the **Add to Master Files** action.

You can view the *master files* for the current resource in the **Properties** dialog of the **Project** view and the *master files* for the current editor in the **Properties** and **Information** views.

Project Validation and Transformation

The **Master Files Support** is also useful for project level validation and transformation. When you hover the cursor over a file in the `Master Files` directory, Oxygen XML Editor displays the **Validate** and **Transform** buttons at the right of the file. Select one of these buttons to run a transformation, or validation scenario. If the current node is selected, Oxygen XML Editor executes a batch transformation and validation. If the current node is not selected, Oxygen XML Editor executes the validation and transformation for the current node only. The behavior of these actions is the same as the behavior of the transformation actions that are available in the contextual menu.



Note: The tooltip of the **Validate** and **Transform** buttons displays the associated scenarios that you can execute.

When you hover the cursor over the `Master Files` directory itself, apart from the **Validate** and **Transform** buttons, Oxygen XML Editor displays a **Help** button. To open the **Help** section regarding the **Master Files Support** click this button, or press **F1** on your keyboard.

The Contextual Menu of the Master Files

The contextual menu of the `Master Files` directory contains the following actions:

New

Allows you to create a **File...**, **Logical Folder...**, or **Project...**

Add Files...

Allows you to add *master files* to the `Master Files` directory.

Add Edited File

Use this option to add the currently edited file to the `Master Files` directory.

Open All Files

Opens all the files of the `Master Files` directory.

Paste

Pastes the files you copy in the `Master Files` directory.

Rename

Allows you to rename a file in the `Master Files` directory.

Refresh

Refreshes the content of the `Master Files` directory.

Find/Replace in Files...

Opens the [Find/Replace dialog box](#).

XPath in Files...

Opens the [XPath/XQuery Builder view](#) that allows you to compose XPath and XQuery expressions and execute them over the currently edited XML document.

Open/Find Resource...

Opens the [Open/Find Resource dialog box](#).

Check Spelling in Files...

Opens the [Check Spelling in Files dialog box](#).

Transform

Provides access to one of the following actions:

Apply Transformations Scenario(s)

Applies the transformation scenarios associated with the `Master Files` directory.

Configure Transformation Scenario(s)...

Opens the [Configure Transformation Scenario dialog box](#).

Transform with...

Opens the **Transform with** dialog box that allows you to select the transformation scenario you want to execute.

Validate

Provides access to one of the following actions:

Check Well-Formedness

Allows you to check if a document is *Namespace Well-Formed XML*.

Validate

Oxygen XML Editor performs the validation of the *master files*.

Validate with Schema...

Opens the **Validate with** dialog box. Oxygen XML Editor performs the validation of the *master files* using a schema.

Configure Validation Scenario(s)...

Opens the [Configure Validation Scenario dialog box](#).

Detect Master Files from Project...

Enables automatic detection of *master files*.

Enable Master Files Support

Select this option to enable the **Master Files Support**.

Editing XML Documents

This section explains the XML editing features of the application. All the user interface components and actions available to users are described in detail with appropriate procedures for various tasks.

Associate a Schema to a Document

This section explains the methods of associating a schema to a document for validation and content completion purposes.

Setting a Schema for Content Completion

This section explains the available methods of setting a schema for content completion in an XML document edited in Oxygen XML Editor.

Supported Schema Types for XML Documents

The supported schema types are:

- W3C XML Schema 1.0 and 1.1 (with and without embedded Schematron rules);
- DTD;
- Relax NG - XML syntax (with and without embedded Schematron rules);
- Relax NG - compact syntax;
- NVDL;
- Schematron (both ISO Schematron and Schematron 1.5).

Setting a Default Schema

When trying to detect a schema, Oxygen XML Editor searches in multiple locations, in the exact following order:

- The *validation scenario* associated with the document.
- The validation scenario associated with the document type (if defined).
- The document schema declaration.

 **Note:** If a DTD schema is specified in the document, the content completion for Author mode is based on this schema (even if there is already one detected from the validation scenario).

- The document type schema definition. Each document type available in *Document Type Association* preferences page contains a set of rules for associating a schema with the current document.

 **Note:** The locations are sorted by priority, from high to low.

The schema has one of the following types: XML Schema, XML Schema with embedded Schematron rules, Relax NG (XML syntax or compact syntax), Relax NG (XML syntax) with embedded Schematron rules, Schematron, DTD, NVDL.

The rules are applied in the order they appear in the table and take into account the local name of the root element, the default namespace and the file name of the document.

Important:

The editor is creating the content completion lists by analysing the specified schema and the current context (the position in the editor). If you change the schema, then the list of tags to be inserted is updated.

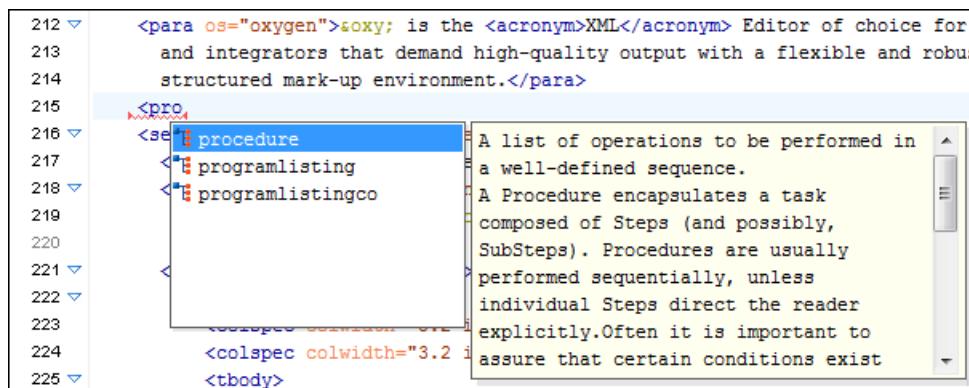


Figure 79: Content Completion Driven by DocBook DTD

Making the Schema Association Explicit in the XML Instance Document

The schema used by the **Content Completion Assistant** and **document validation** engine can be associated with the document using the  **Associate Schema** action. For most of the schema types, it uses *the XML-model processing instruction*, the exceptions being:

- W3C XML Schema - the `xsi:schemaLocation` attribute is used.
- DTD - the DOCTYPE declaration is used.

The association can specify a relative file path or a URL of the schema. The advantage of relative file path is that you can configure the schema at file level instead of document type level.

Select the  **Associate schema** action from the **Document > Schema** menu or the **Document** toolbar to select the schema that will be associated with the XML document. The **Associate Schema** dialog box is displayed:

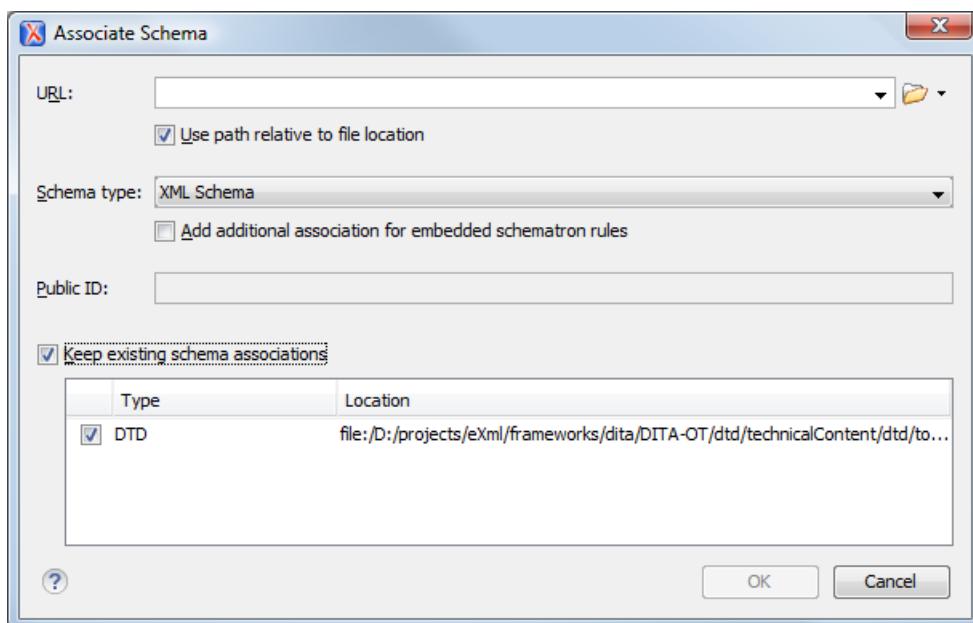


Figure 80: The Associate Schema Dialog Box

The available options are:

- **URL** - Contains a predefined set of schemas that are used more often and it also keeps a history of the last used schemas. The URL must point to the schema file which can be loaded from the local disk or from a remote server through HTTP(S), FTP(S) or a *custom protocol*.
- **Schema type** - Selected automatically from the list of possible types in the **Schema type** combo box (XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, NVDL) based on the extension of the schema file that was entered in the **URL** field.
- **Public ID** - Specify a public ID if you have selected a DTD.
- **Add additional association for embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules, enable this option.
- **Use path relative to file location** - Enable this option if the XML instance document and the associated schema contain relative paths. The location of the schema file is inserted in the XML instance document as a relative file path. This practice allows you, for example, to share these documents with other users, without running into problems caused by different project locations on physical disk.
- **Keep existing schema associations** - Enable this option to keep the associations of the currently edited document with a Schema when you associate a new one.

The association with an XML Schema is added as an attribute of the root element. The **Associate schema** action adds a:

- `xsi:schemaLocation` attribute, if the root element of the document sets a default namespace with an `xmlns` attribute.
- or a `xsi:noNamespaceSchemaLocation` attribute, if the root element does not set a default namespace.

The association with a DTD is added as a DOCTYPE declaration. The association with a Relax NG , Schematron or NVDL schema is added as [xml-model processing instruction](#).

Associating a Schema With the Namespace of the Root Element

The namespace of the root element of an XML document can be associated with an XML Schema using an [XML catalog](#). If there is no `xsi:schemaLocation` attribute on the root element and the [XML](#) document is not matched with a [document type](#), the namespace of the root element is searched in [the XML catalogs set in Preferences](#).

If the XML catalog contains an `uri` or `rewriteUri` or `delegateUri` element, its schema will be used by the application to drive the [content completion](#) and document [validation](#).

The `xml-model` Processing Instruction

The `xml-model` processing instruction associates a schema with the XML document that contains the processing instruction. It must be added at the beginning of the document, just after the XML prologue. The following code snippet contains an `xml-model` processing instruction declaration:

```
<?xml-model href="../schema.sch" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"
phase="ALL" title="Main schema"?>
```

It is available in the [Content Completion Assistant](#), before XML document root element, and includes the following attributes:

- `href` (required) - The schema file location.
- `type` - The content type of the schema. This is an optional attribute with the following possible values for each specified type:
 - DTD - The recommended value is `application/xml-dtd`.
 - W3C XML Schema - The recommended value is `application/xml`, or can be left unspecified.
 - RELAX NG XML Syntax - The recommended value is `application/xml`, or can be left unspecified.
 - RELAX NG Compact Syntax - The recommended value is `application/relax-ng-compact-syntax`.
 - Schematron - The recommended value is `application/xml`, or can be left unspecified.
 - NVDL - The recommended value is `application/xml`, or can be left unspecified.
- `schematypens` - The namespace for the schema language of the referenced schema with the following possible values:
 - DTD - Not specified.
 - W3C XML Schema - The recommended value is `http://www.w3.org/2001/XMLSchema`.
 - RELAX NG XML Syntax - The recommended value is `http://relaxng.org/ns/structure/1.0`.
 - RELAX NG Compact Syntax - Not specified.
 - Schematron - The recommended value is `http://purl.oclc.org/dsdl/schematron`.
 - NVDL - The recommended value is `http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0`.
- `phase` - The phase name for the validation function in Schematron schema. This is an optional attribute. To run all phases from the Schematron, use the special `#ALL` value. If the phase is not specified, the default phase that is configured in the Schematron will be applied.
- `title` - The title for the associated schema. This is an optional attribute.

Older versions of Oxygen XML Editor used the `oxygen` processing instruction with the following attributes:

- `RNGSchema` - Specifies the path to the Relax NG schema that is associated with the current document.
- `type` - Specifies the type of Relax NG schema. It is used along with the `RNGSchema` attribute and can have the value `xml` or `compact`.
- `NVDLSchema` - Specifies the path to the NVDL schema that is associated with the current document.
- `SCHSchema` - Specifies the path to the SCH schema that is associated with the current document.



Note: Documents that use the oxygen processing instruction are compatible with newer versions of Oxygen XML Editor.

Learning Document Structure

When working with documents that do not specify a schema, or for which the schema is not known or does not exist, Oxygen XML Editor is able to learn and translate the document structure to a DTD. You can choose to save the learned structure to a file in order to provide a DTD as an initialization source for *content completion* and *document validation*. This feature is also useful for producing DTD's for documents containing personal or custom element types.

When you open a document that is not associated with a schema, Oxygen XML Editor automatically learns the document structure and uses it for *content completion*. To disable this feature you have to uncheck the checkbox **Learn on open document in the user preferences**.

Create a DTD from Learned Document Structure

When there is no schema associated with an XML document, Oxygen XML Editor can learn the document structure by parsing the document internally. This feature is enabled with *the option Learn on open document* that is available in the user preferences.

To create a DTD from the learned structure:

1. Open the XML document for which a DTD will be created.
2. Go to **Document > XML Document > Learn Structure > Ctrl Shift L (Command Shift L on OS X)**. The **Learn Structure** action reads the mark-up structure of the current document. The **Learn completed** message is displayed in the application's status bar when the action is finished.
3. Go to **Document > XML Document > Save Structure > Ctrl Shift S (Command Shift S on OS X)**. Enter the DTD file path.
4. Press the **Save** button.

Content Completion Assistant

The intelligent **Content Completion Assistant** available in Oxygen XML Editor enables rapid, in-line identification and insertion of structured language elements, attributes and, in some cases, their parameter options.

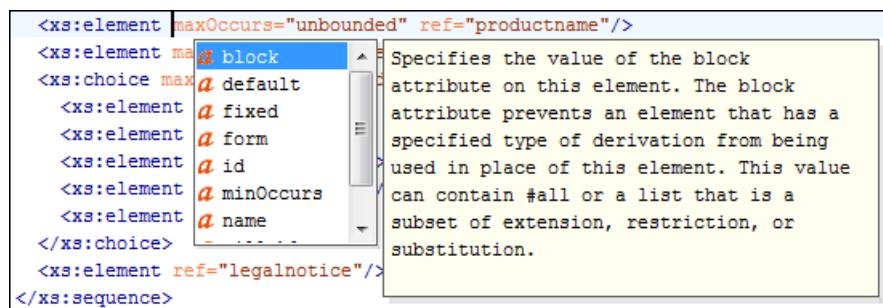


Figure 81: Content Completion Assistant

The functioning of the **Content Completion Assistant** feature is schema-driven (XML Schema, DTD, and RELAX NG). When Oxygen XML Editor detects a schema, it logs its URL in the *Information view*.

The **Content Completion Assistant** is enabled by default. To disable it, *open the Preferences dialog box* and go to **Editor > Content Completion**. It is activated:

- automatically, after a configurable delay from the last key press of the < character. You can adjust the delay *from the Content Completion preferences page*
- on demand, by pressing **Ctrl Space (Command Space on OS X)** on a partial element or attribute name.



Note: If the Content Completion list contains only one valid proposal, when you press the **Ctrl Space (Command Space on OS X)** shortcut key, the proposal is automatically inserted.



Note: You can also start the **Content Completion Assistant** from **Document > Content Completion > Start Content Completion**.

When active, it displays a list of context-sensitive proposals valid at the current caret position. Elements are highlighted in the list using the Up and Down cursor keys on your keyboard. For each selected item in the list, the **Content Completion Assistant** displays a documentation window. You can customize the size of the documentation window by dragging its top, right, and bottom borders.

To insert the selected content:

- press Enter or Tab on your keyboard to insert both the start and end tags.
- press **Ctrl Enter (Command Enter on OS X)** on your keyboard. Oxygen XML Editor inserts both the start and end tags and positions the cursor between the tags, so you can start typing content.



Note: When the DTD, XML Schema or RELAX NG schema specifies required child elements for the newly added element, they are inserted automatically only if the Add Element Content option (found in the [Content Completion preferences page](#)) is enabled. The **Content Completion Assistant** can also add optional content and first choice particle, as specified in the DTD or XML Schema or RELAX NG schema. To activate this feature, [open the Preferences dialog box](#), go to **Content Completion**, and select the **Add optional content** and **Add first Choice particle** check boxes.

After inserting an element, the cursor is positioned:

- before the > character of the start tag, if the element allows attributes, in order to enable rapid insertion of any of the attributes supported by the element. Pressing the space bar displays the Content Completion list once again. This time it contains the list of allowed attribute names. If the attribute supports a fixed set of parameters, the assistant list displays the list of valid parameters. If the parameter setting is user-defined and therefore variable, the assistant is closed to enable manual insertion. The values of the attributes can be learned from the same elements in the current document
- after the > character of the start tag if the element has no attributes.

The **Content Completion Assistant** is displayed:

- anywhere within a tag name or at the beginning of a tag name in an XML document, XML Schema, DTD, or Relax NG (full or compact syntax) schema
- anywhere within an attribute name or at the beginning of an attribute name in any XML document with an associated schema
- within attribute values or at the beginning of attribute values in XML documents where lists of possible values have been defined for that element in the schema associated with the document.

The items that populate the **Content Completion Assistant** depend on the element structure specified in the DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema associated with the edited document.



Note: The **Content Completion Assistant** is able to offer elements defined both by XML Schemas version 1.0 and 1.1.

The number and type of elements displayed by the **Content Completion Assistant** is dependent on the cursor's current position in the structured document. The child elements displayed within a given element are defined by the structure of the specified DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema.

A schema may declare certain attributes as *ID* or *IDREF/IDREFS*. When the document is validated, Oxygen XML Editor checks the uniqueness and correctness of the ID attributes. It also collects the attribute values declared in the document to prepare the **Content Completion Assistant**'s list of proposals. This is available for documents that use DTD, XML Schema, and Relax NG schema.

Also, values of all the *xml:id* attributes are handled as ID attributes. They are collected and displayed by the **Content Completion Assistant** as possible values for *anyURI* attributes defined in the schema of the edited document. This works only for XML Schema and Relax NG schemas.

For documents that use an XML Schema or Relax NG schema, the content assistant offers proposals for attributes and elements values that have as type an enumeration of tokens. Also, if a default value or a fixed value is defined in the

XML Schema used in validation for an attribute or element, then that value is offered in the **Content Completion Assistant** window.

Set Schema for Content Completion

The DTD, XML Schema, Relax NG, or NVDL schema used to populate the **Content Completion Assistant** is specified in the following methods, in order of precedence:

- The schema specified explicitly in the document. In this case Oxygen XML Editor reads the beginning of the document and resolves the location of the DTD, XML Schema, Relax NG schema, or NVDL schema.
- The default schema rule declared in [the Document Type Association preferences panel](#) which matches the edited document.
- For XSLT stylesheets, the schema specified in the Oxygen XML Editor [Content Completion options](#). Oxygen XML Editor will read the Content Completion settings when the prolog fails to provide or resolve the location of a DTD, XML Schema, Relax NG or NVDL schema.
- For XML Schemas, the schema specified in the Oxygen XML Editor [Content Completion options](#). Oxygen XML Editor will read the Content Completion settings and the specified schema will enhance the content completion inside the xs:annotation/xs:appinfo elements of the XML Schema.

Content Completion in Documents with Relax NG Schemas

Inside the documents that use a Relax NG schema, the **Content Completion Assistant** is able to present element values if such values are specified in the Relax NG schema. Also in Relax NG documents the **Content Completion Assistant** presents additional values of type ID for an *anyURI* data type. It presents also pattern names defined in the Relax NG schema as possible values for pattern references. For example if the schema defines an enumValuesElem element like:

```
<element name="enumValuesElem">
  <choice>
    <value>value1</value>
    <value>value2</value>
    <value>value3</value>
  </choice>
</element>
```

In documents based on this schema, the **Content Completion Assistant** offers the following list of values:

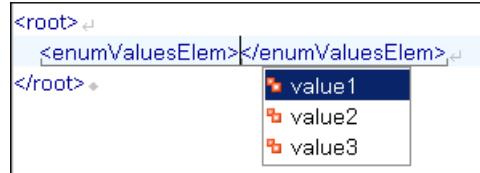


Figure 82: Content Completion assistant - element values in Relax NG documents

Schema Annotations

A schema annotation is a documentation snippet associated with the definition of an element or attribute in a schema. If such a schema is associated with an XML document, the annotations are displayed in:

- the Content Completion Assistant.
- a small tooltip window shown when the mouse hovers over an element or attribute. The tooltip window can be invoked at any time using the F2 shortcut.

The schema annotations support is available the schema type is one of the following: XML Schema, Relax NG, NVDL, or DTD. If you want to turn off this feature, disable the [Show annotations in Content Completion Assistant](#) option.

Styling Annotations with HTML

You can use HTML format in the annotations you add in an XML Schema or Relax NG schema. This improves the visual appearance and readability of the documentation window displayed when editing XML documents validated

against such a schema. An annotation is recognized and displayed as HTML if it contains at least one HTML element, like: `div`, `body`, `p`, `br`, `table`, `ul`, or `ol`.

The HTML rendering is controlled by the **Show annotations using HTML format, if possible** option. When this options is disabled, the annotations are converted and displayed as plain text. If the annotation contains one or more HTML tags (`p`, `br`, `ul`, `li`), they are rendered as an HTML document loaded in a web browser: `p` begins a new paragraph, `br` breaks the current line, `ul` encloses a list of items, `li` encloses an item of the list.

Collecting Annotations from XML Schemas

In an XML Schema the annotations are specified in an `<xs:annotation>` element like this:

```
<xs:annotation>
  <xs:documentation>
    Description of the element.
  </xs:documentation>
</xs:annotation>
```

For XML Schema, if an element or attribute does not have a specific annotation, then Oxygen XML Editor looks for an annotation in the type definition of that element or attribute.

Collecting Annotations from Relax NG Schemas

For Relax NG schema element / attribute annotation are made using the `<documentation>` element from the <http://relaxng.org/ns/compatibility/annotations/1.0> namespace. However, any element outside the Relax NG namespace (<http://relaxng.org/ns/structure/1.0>) is handled as annotation and the text content is displayed in the annotation window. To activate this behaviour, enable the ***Use all Relax NG annotations as documentation*** option.

Collecting Annotation from DTDs

For DTD Oxygen XML Editor defines a custom mechanism for annotation using comments enabled from the option ***Use DTD comments as annotations***. Following is an example of a DTD annotation:

```
<!--doc:Description of the element. -->
```

Content Completion Helper Views

Information about the current element being edited is also available in the Model view and Attributes view, located by default on the left-hand side of the main window. The Model view and the Attributes view combined with the powerful Outline view provide spatial and insight information on the edited document.

The Model View

The **Model** view presents the structure of the currently edited tag and tag documentation defined as annotation in the schema of the current document.

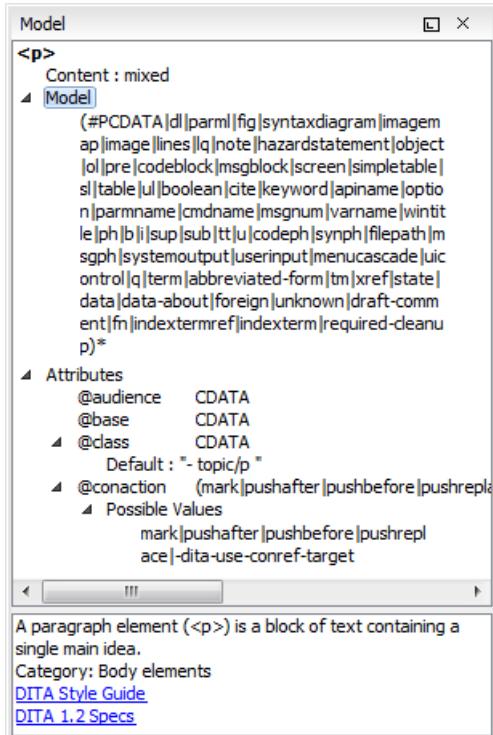


Figure 83: The Model View

The **Model** view is comprised of:

- *An element structure panel.*
- *An annotation panel.*

The Element Structure Panel

The element structure panel shows the structure of the current edited or selected tag in a tree-like format.

The information includes the name, model and attributes the currently edited tag may have. The allowed attributes are shown along with imposed restrictions, if any.

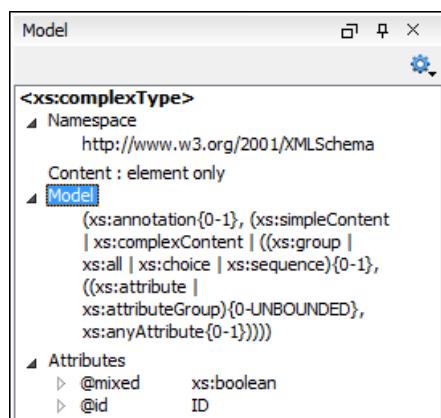


Figure 84: The Element Structure Panel

The Annotation Panel

The **Annotation** panel displays the annotation information for the currently selected element. This information is collected from the XML schema.

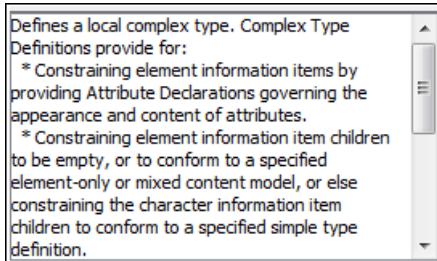


Figure 85: The Annotation panel

The Attributes View

The **Attributes View** presents all possible attributes of the current element.

The view allows you to insert attributes or change the value of the already used attributes for the current editable element. An element is editable if either one of the following is true:

- the CSS stylesheet associated with the document does not specify a **false** value for the *-oxy-editable* property associated with the element.
- the element is entirely included into a deleted *Track Changes* marker.
- the element is part of a content fragment that is referenced in **Author** mode from another document.

The attributes present in the document are rendered bold in the **Attributes View**. You can start editing the value of an attribute by clicking the **Value** cell of a table row. If the possible values of the attribute are specified as list in the schema associated with the edited document, the **Value** cell works as a list box from which you can select one of the possible values to be inserted in the document.

The **Attributes** table is sortable, three sorting modes being available by clicking the **Attribute** column name: alphabetically ascending, alphabetically descending, or custom order. The custom order places the already used attributes at the beginning of the table, as they appear in the element, followed by the rest of the allowed elements, as they are declared in the associated schema.

Attributes	
xs:element [http://www.w3.org/2001/XMLSchema]	
Attribute	Value
abstract	false
block	false
default	true
final	
fixed	
id	
name	email
nilable	false
substitutionGroup	
type	xs:string

Figure 86: The Attributes View

The Elements View

The Elements view presents a list of all defined elements that you can insert at the current caret position according to the schema associated to the document. Double-clicking any of the listed elements inserts that element in the edited document. All elements from a sequence are presented but the invalid proposals (which cannot be inserted in the current context) are grayed-out.



Figure 87: The Elements View

The Entities View

This view displays a list with all entities declared in the current document, as well as built-in ones. Double-clicking one of the entities will insert it at the current cursor position. You can also sort entities by name and value by clicking the column headers.

Name	Value
lt	<
gt	>
amp	&
apos	'
quot	"
abbrev-d-att	(topic abbrev-d)
concept-att	(topic concept)
hazard-d-att	(topic hazard-d)
hi-d-att	(topic hi-d)
included-domains	&concept-att; ...
indexing-d-att	(topic indexing-d)
nbspace	
pr-d-att	(topic pr-d)
sw-d-att	(topic sw-d)
ui-d-att	(topic ui-d)
ut-d-att	(topic ut-d)

Figure 88: The Entities View

The view features a filtering capability that allows you to search an entity by name, value, or both. Also, you can choose to display the internal or external entities.

 **Note:** When entering filters, you can use the ? and * wildcards. Also, you can enter multiple filters by separating them with comma.

Code Templates

Code templates are code fragments that can be inserted quickly at the current editing position . Oxygen XML Editor comes with a set of built-in code templates for CSS, LESS, Schematron, XSL, XQuery, and XML Schema document types. You can also [define your own code templates and share them with others](#).

To get a complete list of available code templates, press **Ctrl Shift Space (Command Shift Space on OS X)** in **Text** mode or **Enter** in **Author** mode. To enter the code template, select it from the list or type its code and press **Enter**. If a shortcut key has been assigned to the code template, you can also use the shortcut key to enter it. Code templates are displayed with a  symbol in the content completion list.

When the **Content Completion Assistant** is invoked (**Ctrl Space (Command Space on OS X)**), it also presents a list of code templates specific to the type of the active editor.

To watch our video demonstration about code templates, go to http://oxygenxml.com/demo/Code_Templates.html.

Configuring the Proposals in the Content Completion Assistant

Oxygen XML Editor gathers information from the associated schemas (DTDs, XML Schema, RelaxNG) to determine the proposals that appear in the **Content Completion Assistant**. Oxygen XML Editor also includes support that allows you to configure the possible attribute or element values for the proposals. To do so, a configuration file can be used, along with the associated schema, to add or replace possible values for attributes or elements that are proposed in the **Content Completion Assistant**. An example of a specific use-case is if you want the **Content Completion Assistant** to propose several possible values for the language code whenever you use an `xml:lang` attribute.

To configure content completion proposals, follow these steps:

1. Create a new `resources` folder (if it does not already exist) in the frameworks directory for the document type. For instance: `OXYGEN_INSTALL_DIR/frameworks/dita/resources`.
2. *Open the Preferences dialog box* and go to **Document Type Association**. Edit the document type configuration for your XML vocabulary, and in the **Classpath** tab add a link to that `resources` folder.
3. Use the **New** document wizard to create a configuration file using the **Content Completion Configuration** file template.
4. Make the appropriate changes to your custom configuration file. The file template includes details about how each element and attribute is used in the configuration file.
5. Save the file in the `resources` folder, using the fixed name: `cc_value_config.xml`.
6. Re-open the application and open an XML document. In the **Content Completion Assistant** you should see your customizations.

The Configuration File

The configuration file is composed of a series of `match` instructions that will match either an element or an attribute name. A new value is specified inside one or more `item` elements, which are grouped inside an `items` element. The behavior of the `items` element is specified with the help of the `action` attribute, which can have any of the following values:

- `append` - Adds new values to appear in the proposals list (default value).
- `addIfEmpty` - Adds new values to the proposals list, only if no other values are contributed by the schema.
- `replace` - Replaces the values contributed by the schema with new values to appear in the proposals list.

The values in the configuration file can be specified either directly or by calling an external XSLT file that will extract data from any external source.

Example - Specifying Values Directly

```
<!-- Replaces the values for an element with the local name "lg", from the given namespace -->
<match elementName="lg" elementNS="http://www.oxygenxml.com/ns/samples">
  <items action="replace">
    <item value="stanza"/>
    <item value="refrain"/>
  </items>
</match>

<!-- Adds two values for an attribute with the local name "type", from any namespace -->
<match attributeName="type">
  <items>
    <item value="stanza"/>
    <item value="refrain"/>
  </items>
</match>
```

Example - Calling an External XSLT Script

```
<xslt href="../xsl/get_values_from_db.xsl" useCache="false" action="replace"/>
```

In this example, the `get_values_from_db.xsl` is executed in order to extract values from a database.



Note: A comprehensive XSLT sample is included in the **Content Completion Configuration** file template.

Validating XML Documents

The W3C XML specification states that a program should not continue to process an XML document if it finds a validation error. The reason is that XML software should be easy to write, and that all XML documents should be compatible. With HTML it was possible to create documents with lots of errors (like when you forget an end tag). One of the main reasons that HTML browsers are so big and incompatible, is that they have their own ways to figure out what a document should look like when they encounter an HTML error. With XML this should not be possible.

However, when creating an XML document, errors are very easily introduced. When working with large projects or many files, the probability that errors will occur is even greater. Determining that your project is error-free can be time consuming and even frustrating. For this reason Oxygen XML Editor provides functions that enable easy error identification and rapid error location.

Checking XML Well-formedness

A *Well-Formed XML* document is a document that conforms to the XML syntax rules. A *Namespace Well-Formed XML* document is a document that is XML Well-Formed and is also namespace-wellformed and namespace-valid.

The XML Syntax rules for Well-Formed XML are:

- All XML elements must have a closing tag.
- XML tags are case-sensitive.
- All XML elements must be properly nested.
- All XML documents must have a root element.
- Attribute values must always be quoted.
- With XML, white space is preserved.

The namespace-wellformed rules are:

- All element and attribute names contain either zero or one colon.
- No entity names, processing instruction targets, or notation names contain any colons.

The namespace-valid rules are:

- The prefix *xml* is by definition bound to the namespace name *http://www.w3.org/XML/1998/namespace*. It MAY, but need not, be declared, and MUST NOT be undeclared or bound to any other namespace name. Other prefixes MUST NOT be bound to this namespace name.
- The prefix *xmlns* is used only to declare namespace bindings and is by definition bound to the namespace name *http://www.w3.org/2000/xmlns/*. It MUST NOT be declared or undeclared. Other prefixes MUST NOT be bound to this namespace name.
- All other prefixes beginning with the three-letter sequence *x, m, l*, in any case combination, are reserved. This means that users SHOULD NOT use them except as defined by later specifications and processors MUST NOT treat them as fatal errors.
- The namespace prefix, unless it is *xml* or *xmlns*, MUST have been declared in a namespace declaration attribute in either the start-tag of the element where the prefix is used or in an ancestor element (i.e. an element in whose content the prefixed markup occurs). Furthermore, the attribute value in the innermost such declaration MUST NOT be an empty string.

To check if a document is *Namespace Well-Formed XML*, select the **Check Well-Formedness (Ctrl Shift W)** (**Command Shift W on OS X**) action from the **Document > Validate** menu or from the Validation toolbar drop-down list. If any error is found the result is returned to the message panel. Each error is one record in the result list and is accompanied by an error message. Clicking the record will open the document containing the error and highlight its approximate location.

A not Well-Formed XML Document

```
<root><tag></root>
```

When **Check Well-Formedness** is performed the following error is raised:

```
The element type "tag" must be terminated by the matching end-tag "</tag>"
```

To resolve the error, click in the result list record which will locate and highlight the errors approximate position. Identify which start tag is missing an end tag and insert `</tag>`.

A not namespace-wellformed document

```
<x::y></x::y>
```

When **Check document form** is performed the following error is raised:

```
Element or attribute do not match QName production:  
QName ::= (NCName ':' )?NCName .
```

A not namespace-valid document

```
<x:y></x:y>
```

When **Check document form** is performed the following error is raised:

```
The prefix "x" for element "x:y" is not bound.
```

Also the selected files in the current project can be checked for well-formedness with a single action by selecting the  **Check Well-Formedness** action from the **Validate** submenu when invoking the contextual menu in the [Project view](#).

Validating XML Documents Against a Schema

A *Valid* XML document is a *Well-Formed* XML document that also conforms to the rules of a schema that defines the legal elements of an XML document. The schema type can be: XML Schema, Relax NG (full or compact syntax), Schematron, Document Type Definition (DTD), or Namespace-based Validation Dispatching Language (NVDL).

The purpose of the schema is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

The  **Validate** function ensures that your document is compliant with the rules defined by an associated DTD, XML Schema, Relax NG, or Schematron schema. XML Schema or Relax NG Schema can embed Schematron rules. For Schematron validations you can select the validation phase.

Marking Validation Errors and Warnings

A line with a validation error or warning will be marked in the editor panel by underlining the error region with a red color. Also a red sign will mark the position in the document of that line on the right side ruler of the editor panel. The same will happen for a validation warning, only the color will be yellow instead of red.

The ruler on the right side of the document is designed to display the errors and warnings found during the validation process and also to help the user to locate them more easily. The ruler contains the following areas:

- Top area containing a success validation indicator that will turn green in case the validation succeeded or red otherwise.
A more detailed report of the errors is displayed in the tooltip of the validation indicator. In case there are errors, only the first three of them will be presented in the tooltip.

- Middle area where the error markers are depicted in red (with a darker color tone for the current selected one). To limit the number of markers shown [open the Preferences dialog box](#) and go to **Editor > Document checking > Maximum number of problems reported per document.**

Clicking on a marker will highlight the corresponding text area in the editor. The error message is displayed both in the tool tip and in the error area on the bottom of the editor panel.

The **Document checking user preferences** are easily accessible from the button displayed at the beginning of the error message on the bottom of the editor panel.

- Bottom area containing two navigation arrows that will go to the next or to the previous error and a button for clearing all the error markers from the ruler. The same actions can be triggered from menu **Document > Automatic validation > Next Error Ctrl . (Command . on OS X)** and **Document > Automatic validation > Previous Error Ctrl , (Command , on OS X)**.

The validation status area is the line at the bottom of the editor panel that presents the message of the current validation error selected on the right side ruler. Clicking on the **Document checking options** button opens the [document checking](#) page in Oxygen XML Editor user preferences.

Status messages from every validation action are logged into the [Information view](#).

If you want to see all the validation error messages [grouped in a view](#) you should use the **Validate** action from the **Document > Validate** menu or from the **Validation** toolbar drop-down list. This action collects all error messages in the **Errors** view.

Customising Assert Error Messages

To customise the error messages that the Xerces or Saxon validation engines display for the `assert` and `assertion` elements, set the `message` attribute on these elements. For Xerces, the `message` attribute has to belong to the `http://xerces.apache.org` namespace. For Saxon, the `message` attribute has to belong to the `http://saxon.sourceforge.net/` namespace. The value of the `message` attribute is the error message displayed in case the assertion fails.

Validation Example - A DocBook Validation Error

In the following DocBook 4 document the content of the `listitem` element does not match the rules of the DocBook 4 schema, that is `docbookx.dtd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.docbook.org/xml/4.4/docbookx.dtd">
<article>
  <title>Article Title</title>
  <sect1>
    <title>Section1 Title</title>
    <itemizedlist>
      <listitem>
        <link>a link here</link>
      </listitem>
    </itemizedlist>
  </sect1>
</article>
```

The **Validate Document** action will return the following error:

```
Unexpected element "link". The content of the parent element type must match
"(calloutlist|glosslist|bibliolist|itemizedlist|orderedlist|segmentedlist|simplelist
|variablelist|caution|important|note|tip|warning|literallayout|programlisting
|programlistingco|screen|screenco|screenshot|synopsis|cmdsynopsis|funcsynopsis
|classsynopsis|fieldsynopsis|constructorsynopsis|destructorsynopsis|methodsynopsis
|formalpara|para|simpara|address|blockquote|graphic|graphicco|mediaobject|mediaobjectco
|informalequation|informalexample|informalfigure|informaltable|equation|example|figure
|table|msgset|procedure|sidebar|qandaset|task|anchor|bridgehead|remark|highlights
|abstract|authorblurb|epigraph|indexterm|beginpage)+".
```

This error message is a little more difficult to understand, so understanding of the syntax or processing rules for the DocBook XML DTD's `listitem` element is recommended. However, the error message does give us a clue as to the source of the problem, indicating that “The content of element type c must match”.

Luckily most standards based DTD's, XML Schema's and Relax NG schemas are supplied with reference documentation. This enables us to lookup the element and read about it. In this case you should learn about the child elements of `listitem` and their nesting rules. Once you have correctly inserted the required child element and nested it in accordance with the XML rules, the document will become valid on the next validation test.

Automatic Validation

Oxygen XML Editor *can be configured* to mark validation errors in the document as you are editing. If you *enable the Automatic validation option* any validation errors and warnings will be *highlighted automatically in the editor panel*. The automatic validation starts parsing the document and marking the errors after a *configurable delay* from the last key typed. Errors are highlighted with underline markers in the main editor panel and small rectangles on the right side ruler of the editor panel, *in the same way as for manual validation invoked by the user*.

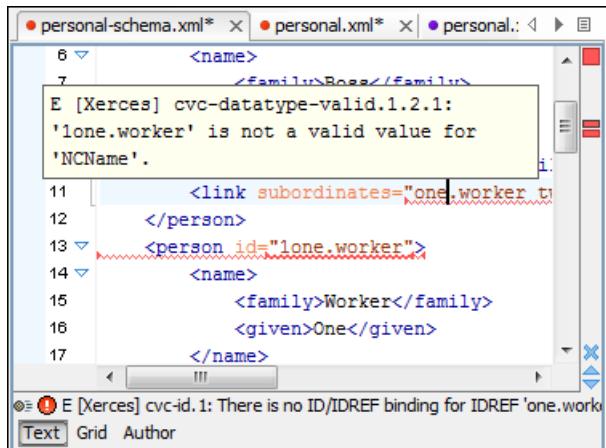


Figure 89: Automatic Validation on the Edited Document

If the error message is long and it is not displayed completely in the error line at the bottom of the editing area, double-clicking on the error icon at the left of the error line or on the error line displays an information dialog box with the full error message. The arrow buttons of the dialog box enable the navigation to other errors issued by the Automatic Validation feature.

Custom Validators

If you need to validate the edited document with a validation engine that is different from the built-in engine, you can configure external validators in the Oxygen XML Editor preferences. After a custom validation engine is *properly configured*, it can be applied on the current document by selecting it from the list of custom validation engines in the **Validation** toolbar drop-down list. The document is validated against the schema declared in the document.

Some validators are configured by default but there are third party processors which do not support the *output message format* of Oxygen XML Editor for linked messages:

- **LIBXML** - Included in Oxygen XML Editor (Windows edition only). It is associated to XML Editor. It is able to validate the edited document against XML Schema, Relax NG schema full syntax, internal DTD (included in the XML document) or a custom schema type. XML catalogs support (the `--catalogs` parameter) and XInclude processing (`--xincl`) are enabled by default in the preconfigured LIBXML validator. The `--postvalid` parameter is also set by default which allows LIBXML to validate correctly the main document even if the XInclude fragments contain IDREFS to ID's located in other fragments.

For validation against an external DTD specified by URI in the XML document, add the `--dtdvalid ${ds}` parameter manually to the DTD validation command line. `${ds}` represents the detected DTD declaration in the XML document.

Caution: File paths containing spaces are not handled correctly in the LIBXML processor. For example the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled by LIBXML if Oxygen XML Editor is installed in the default location on Windows (C:\Program Files) because the built-in XML catalog files are stored in the frameworks subfolder of the installation folder which in this case contains at least one space character in the file path.



Attention: On OS X if the full path to the LIBXML executable file is not specified in the **Executable path** text field, some errors may occur during validation against a W3C XML Schema, such as:

```
Unimplemented block at ... xmlschema.c
```

To avoid these errors, specify the full path to the LIBXML executable file.

- **Saxon SA** - Included in Oxygen XML Editor. It is associated to XML Editor and XSD Editor. It is able to validate XML Schema schemas and XML documents against XML Schema schemas. The validation is done according to the W3C XML Schema 1.0 or 1.0. This can be *configured in Preferences*.
- **MSXML 4.0** - Included in Oxygen XML Editor (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
- **MSXML.NET** - Included in Oxygen XML Editor (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
- **XSV** - Not included in Oxygen XML Editor. Windows and Linux distributions of XSV can be downloaded from <http://www.cogsci.ed.ac.uk/~ht/xsv-status.html>. The executable path is *already configured in Oxygen XML Editor* for the [OXYGEN_DIR] / xsv installation folder. If it is installed in a different folder the predefined executable path must be *corrected in Preferences*. It is associated to XML Editor and XSD Editor. It is able to validate the edited document against XML Schema or a custom schema type.
- **SQC (Schema Quality Checker from IBM)** - Not included in Oxygen XML Editor. It can be downloaded *from here* (it comes as a .zip file, at the time of this writing SQC2.2.1.zip is about 3 megabytes). The executable path and working directory are already configured for the SQC installation directory [OXYGEN_DIR] / sqc. If it is installed in a different folder the predefined executable path and working directory must be *corrected in the Preferences page*. It is associated to XSD Editor.

A custom validator cannot be applied on files loaded through an *Oxygen XML Editor custom protocol plugin* developed independently and added to Oxygen XML Editor after installation.

Linked Output Messages of an External Engine

Validation engines display messages in an output view at the bottom of the Oxygen XML Editor window. If such an output message (*warning*, *error*, *fatal error*, etc) spans between three to six lines of text and has the following format, then the message is linked to a location in the validated document. A click on the message in the output view highlights the location of the message in an editor panel containing the file referenced in the message. This behavior is similar to the linked messages generated by the default built-in validator.

Linked messages have the following format:

- *Type*:[F|E|W] (the string *Type*: followed by a letter for the type of the message: fatal error, error, warning) - this property is optional in a linked message
- *SystemID*: a system ID of a file (the string *SystemID*: followed by the system ID of the file that will be opened for highlighting when the message is clicked in the output message - usually the validated file, the schema file or an included file)
- *Line*: a line number (the string *Line*: followed by the number of the line that will be highlighted)
- *Column*: a column number (the string *Column*: followed by the number of the column where the highlight will start on the highlighted line) - this property is optional in a linked message
- *EndLine*: a line number (the string *EndLine*: followed by the number of the line where the highlight ends) - this property is optional in a linked message
- *EndColumn*: a column number (the string *EndColumn*: followed by the number of the column where the highlight ends on the end line) - this property is optional in a linked message



Note: The *Line/Column* pair works in conjunction with the *EndLine/EndColumn* pair. Thus, if both pairs are specified, then the highlight starts at *Line/Column* and ends at *EndLine/EndColumn*. If the *EndLine/EndColumn* pair is missing, the highlight starts from the beginning of the line identified by the *Line* parameter and ends at the column identified by the *Column* parameter.

- *AdditionalInfoURL*: the URL string pointing to a remote location where additional information about the error can be found - this line is optional in a linked message.
- *Description*: message content (the string *Description*: followed by the content of the message that will be displayed in the output view).

Example of how a custom validation engine can report an error using the format specified above:

```
Type: E
SystemID: file:///c:/path/to/validatedFile.xml
Line: 10
Column: 20
EndLine: 10
EndColumn: 35
AdditionalInfoURL: http://www.host.com/path/to/errors.html#errorID
Description: custom validator message
```

Validation Scenario

A complex XML document is split in smaller interrelated modules. These modules do not make much sense individually and cannot be validated in isolation due to interdependencies with other modules. Oxygen XML Editor validates the main module of the document when an imported module is checked for errors.

A typical example is the chunking DocBook XSL stylesheet which has `chunk.xsl` as the main module and `param.xsl`, `chunk-common.xsl`, and `chunk-code.xsl` as imported modules. `param.xsl` only defines XSLT parameters. The module `chunk-common.xsl` defines an XSLT template with the name `chunk`. `chunk-code.xsl` calls this template. The parameters defined in `param.xsl` are used in the other modules without being redefined.

Validating `chunk-code.xsl` as an individual XSLT stylesheet, generates misleading errors in regards to parameters and templates that are used but undefined. These errors are only caused by ignoring the context in which this module is used in real XSLT transformations and in which it is validated. To validate such a module, define a validation scenario to set the main module of the stylesheet and the validation engine used to find the errors. Usually this engine applies the transformation during the validation process to detect the errors that the transformation generates.

You can validate a stylesheet with several engines to make sure that you can use it in different environments and have the same results. For example an XSLT stylesheet is applied with Saxon 6.5, Xalan and MSXML 4.0 in different production systems.

Other examples of documents which can benefit of a validation scenario are:

- A complex XQuery with a main module which imports modules developed independently but validated in the context of the main module of the query. In an XQuery validation scenario the default validator of Oxygen XML Editor (Saxon 9) or any connection to a database that supports validation (Berkeley DB XML Database, eXist XML Database, Documentum xDb (X-Hive/DB) 10 XML Database, MarkLogic version 5 or newer) can be set as a validation engine.
- An XML document in which the master file includes smaller fragment files using XML entity references.



Note: When you validate a document for which a master file is defined, Oxygen XML Editor uses the scenarios defined in [the Master Files directory](#).

To watch our video demonstration about how to use a validation scenario in Oxygen XML Editor, go to http://oxygenvml.com/demo/Validation_Scenario.html.

How to Create a Validation Scenario

Follow these steps for creating a validation scenario:

1. To open the **Configure Validation Scenario** dialog box, select the **Configure Validation Scenario(s)...** from the **Document > Validate** menu or the **Validation** toolbar drop-down list.
The **Configure Validation Scenario(s)** dialog box is displayed. It contains the following types of scenarios:
 - **Predefined** scenarios are organized in categories depending on the type of file they apply to. You can identify **Predefined** scenarios by a yellow key icon that marks them as *read-only*. If the predefined scenario is the default scenario of the framework, its name is written in bold font. If you try to edit one of these scenarios, Oxygen XML Editor creates a customizable duplicate.

- **User defined** scenarios are organized under a single category, but you can use the drop-down option box to filter them by the type of file they validate.
-  **Note:** The default validation scenarios are not displayed in the scenarios list. If the current file has no associated scenarios, the preview area displays a message to let you know that you can apply the default validation.

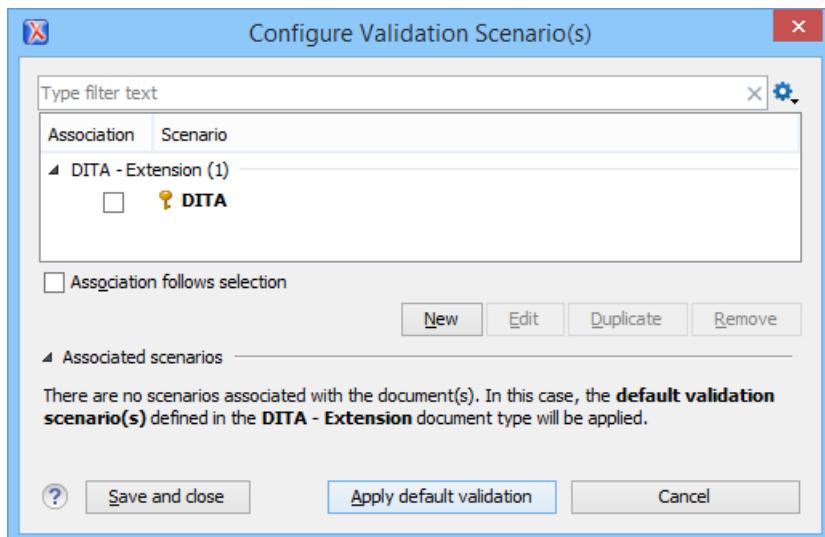


Figure 90: Configure Validation Scenario

2. Press the  New button to add a new scenario.
The New scenarios dialog box that lists all validation units of the scenario is opened.

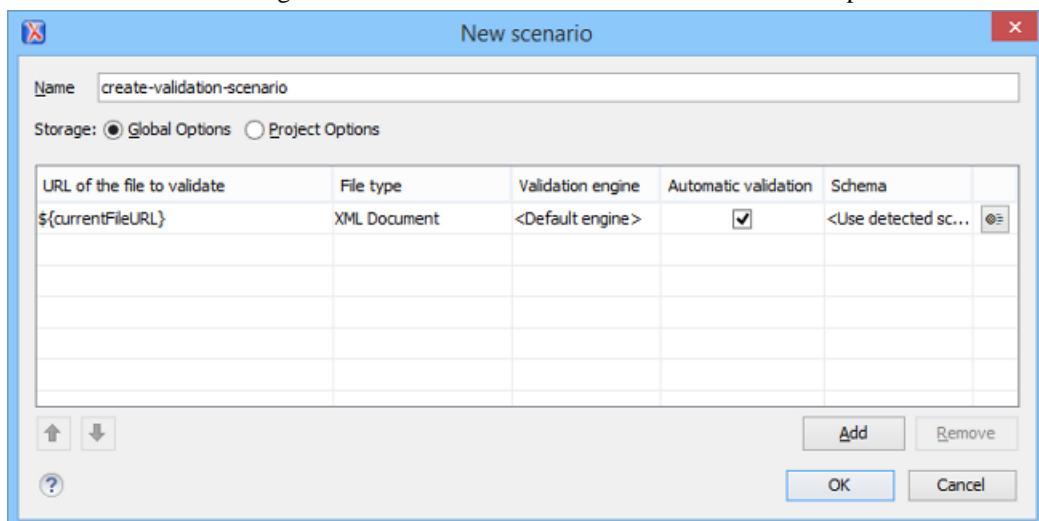


Figure 91: Add / Edit a Validation Unit

The table includes the following information:

- **URL of the file to validate** - The URL of the main module that includes the current module. It is also the entry module of the validation process when the current one is validated.
- **File type** - The type of the document that is validated in the current validation unit. Oxygen XML Editor automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Editor for validation of the type of document to which the current module belongs. **Default engine** is the default setting and it means that the default engine

executes the validation. This engine is set in the **Preferences** pages for the current document type (XML document, XML Schema, XSLT stylesheet, XQuery file, etc.) instead of a validation scenario.

- **Automatic validation** - If this option is checked, the validation operation defined by this row is also applied by *the automatic validation feature*. If the **Automatic validation** feature is *disabled in Preferences*, then this option is ignored, as the Preference setting has a higher priority.
- **Schema** - This option becomes active when you set the **File type** to **XML Document**.
- **Settings** - Opens the **Specify Schema** dialog box that allows you to set a schema for validating XML documents, or a list of extensions for validating XSL or XQuery documents. You can also set a default phase for validation with a Schematron schema.

3. Press the **Add** button to add a new validation unit with default settings.
4. To edit the URL of the main validation module, double-click on its cell in the **URL of the file to validate** column.

Specify the URL of the main module by doing one of the following:

- Use the **Browse** drop-down button to browse for a local, remote, or archived file.
- Use the **Insert Editor Variable** button to insert an *editor variable* or a *custom editor variable*.

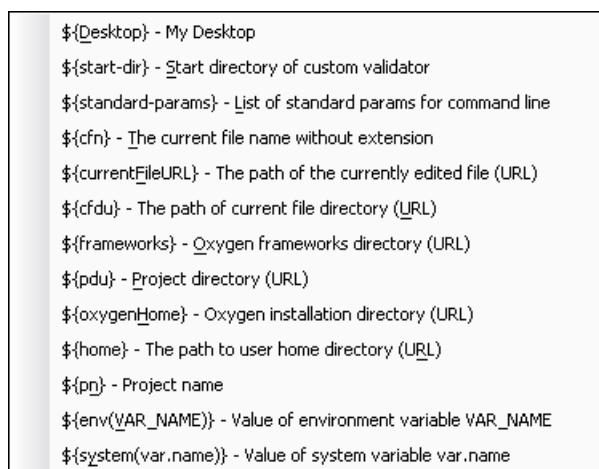


Figure 92: Insert an Editor Variable

5. Select the type of the validated document.

Note that this determines the list of possible validation engines.

6. Select the validation engine.
7. Select the **Automatic validation** option if you want to validate the current unit when the *automatic validation feature is enabled in the Preferences*.
8. Choose the schema to be used during validation (the schema detected after parsing the document or a custom one).

Sharing Validation Scenarios

Sometimes a group of users want to apply the same validation settings, like the main module where the validation starts, the validation engine, the schema, extensions of the engine. In order to apply the same settings consistently it is preferable to share the validation scenario with the settings by storing it at project level and sharing the project file using a source version control system (like CVS, SVN, Source Safe).

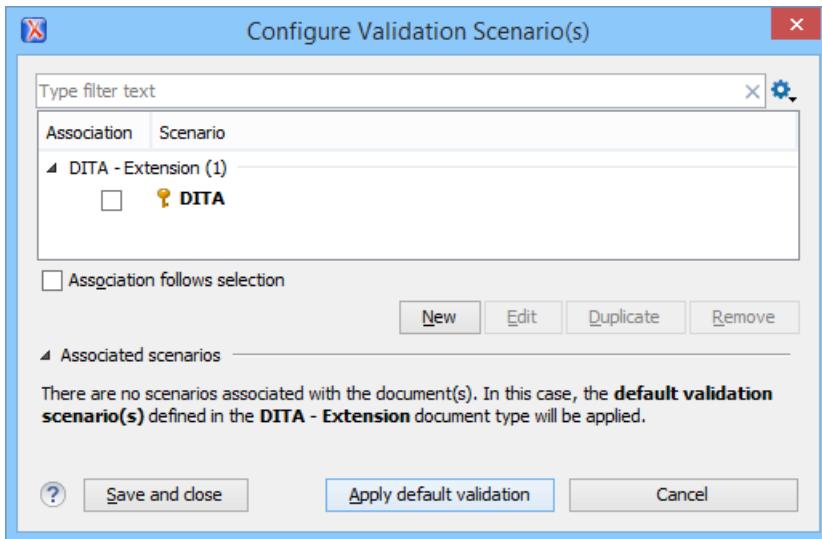


Figure 93: Configure Validation Scenario

You can specify that you want to store a scenario at project level by selecting the **Project Scenarios** option, or you can store them in the user home directory by selecting **Global Scenarios**. When you create a scenario at the project level, the URLs from the scenario become relative to the project URL.

Validation Actions in the User Interface

To validate the currently edited document, use one of the following methods:

- Select the **Validate (Ctrl Shift V (Command Shift V on OS X))** action from the **Document > Validate** menu, from the **Validation** toolbar drop-down list, or from the **Validate** submenu when invoking the contextual menu in the **Project** view. An error list is presented in the message panel. Markup of current document is checked to conform with the specified DTD, XML Schema, or Relax NG schema rules. This action also re-parses the XML catalogs and resets the schema used for content completion.
- Select the **Validate (cached)** action from the **Document > Validate** menu or from the **Validation** toolbar drop-down list. This action caches the schema, allowing it to be reused for the next validation. Markup of the current document is checked to conform with the specified DTD, XML Schema, or Relax NG schema rules.
- **Note:** Automatic validation also caches the associated schema.
- Select the **Validate with...** action from the **Document > Validate** menu, from the **Validation** toolbar drop-down list, or from the **Validate** submenu when invoking the contextual menu in the **Project** view. You can use this action to validate the current document using a schema of your choice (XML Schema, DTD, Relax NG, NVDL, Schematron schema), other than the associated one. An error list is presented in the message panel. Markup of current document is checked to conform with the specified schema rules.
- **Note:** The **Validate with...** action does not work for files loaded through an *Oxygen XML Editor custom protocol plugin* developed independently and added to Oxygen XML Editor after installation.
- Select **Validate with Schema...** from the **Validate** submenu when invoking contextual menu in the **Project** view to choose a schema and validate all selected files with it.

To open the schema used for validating the current document, select the **Open Associated Schema** action from the **Document > Schema** menu.

The **Validation options** button, available in the **Document > Validate** menu, allows you to quickly access to the *validation options* for the built-in validator in the Oxygen XML Editor preferences page.

- Tip:** If a large number of validation errors are detected and the validation process takes too long, you can *limit the maximum number of reported errors in the Preferences page*.

References to XML Schema Specification

If validation is done against XML Schema Oxygen XML Editor indicates a specification reference relevant for each validation error. The error messages contain an **Info** field that when clicked will open the browser on the *XML Schema Part 1: Structures* specification at exactly the point where the error is described. This allows you to understand the reason for that error.

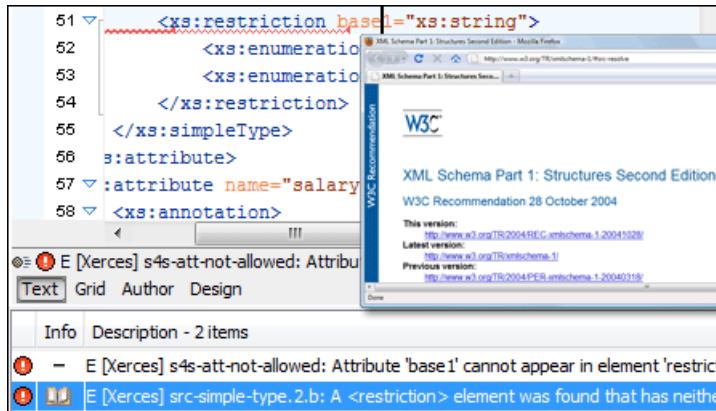


Figure 94: Link to Specification for XML Schema Errors

Resolving References to Remote Schemas with an XML Catalog

When a reference to a remote schema must be used in the validated XML document for interoperability purposes, but a local copy of the schema should be actually used for validation for performance reasons, the reference can be resolved to the local copy of the schema with an *XML catalog*. For example, if the XML document contains a reference to a remote schema `docbook.rng` like this:

```
<?xml-model href="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng" type="application/xml"
schematypens="http://relaxng.org/ns/structure/1.0"?>
```

it can be resolved to a local copy with a catalog entry:

```
<uri name="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng"
uri="rng/docbook.rng"/>
```

An XML catalog can be used also to map a W3C XML Schema specified with an URN in the `xsi:schemaLocation` attribute of an XML document to a local copy of the schema. For example, if the XML document specifies the schema with:

```
<topic xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1">
```

the URN can be resolved to a local schema file with a catalog entry like:

```
<uri name="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1"
uri="topic.xsd"/>
```

Document Navigation

This section explains various methods for navigating the edited XML document.

Quick Document Navigation Using Bookmarks

By using bookmarks, you can mark positions in an edited document so that you can return to it later. This is especially helpful for navigating through large documents or while editing multiple documents. You can place up to nine distinct bookmarks in any document. Shortcut keys are available to place the bookmarks or to return to any of the marked positions. To configure these shortcut keys, go to **Options > Menu Shortcut Keys**.

```

<xsl:output omit-xml-declaration="yes" method="html"/> ↴
<xsl:template match="/" ↴
  <xsl:apply-templates/> ↴
</xsl:template> ↴
<xsl:template match="anchor"> ↴
  <a name="{@target}"> ↴
</xsl:template> ↴
<xsl:template match="articledescription"> ↴
  <table width="100%" border="0" cellpadding="0" cellspacing="0">
    
```

Figure 95: Editor Bookmarks

A bookmark can be inserted by one of the following:

- Right-click on the vertical strip to the left of the editor.
- Select the **Create Bookmark (F9)** action from the **Edit > Bookmarks** menu.

A bookmark can be removed by right-clicking its icon on the vertical strip and select **Remove** or **Remove all (Ctrl+F7)** (you can also find the **Remove all (Ctrl+F7)** action in the **Edit > Bookmarks** menu).

You can navigate the bookmarks by using one of the actions available on the **Edit > Bookmarks > Go to** menu or by using the shortcut keys.

Folding of the XML Elements

An XML document is organized as a tree of elements. When working on a large document you can collapse some elements leaving in the focus only the ones you need to edit. Expanding and collapsing works on individual elements: expanding an element leaves the child elements unchanged.

```

▶ <xsl:template match="articledescription"> ↴ [28 lines]
▼ <xsl:template match="articledescriptions"> ↴
  <xsl:apply-templates/> ↴
</xsl:template> ↴
▼ <xsl:template match="code"> ↴
  <ul> ↴
  ▼ <p class="textSmall"> ↴
▶     <xsl:for-each select="coderow"> ↴ [2 lines]
      </p> ↴
    </ul> ↴
</xsl:template> ↴

```

Figure 96: Folding of the XML Elements

An unique feature of Oxygen XML Editor is the fact that the folds are persistent: the next time you will open the document the folds are restored to the last state so you won't have to collapse the uninteresting parts again.

To toggle the folded state of an element click on the special mark displayed in the left part of the document editor next to the start tag of that element or click on the action **Toggle fold** available from the contextual menu or from the menu **Document > Folding > Toggle fold**. The element extent is marked with a grey line displayed in the left part of the edited document. The grey line always covers the lines of text comprised between the start tag and end tag of the element where the cursor is positioned.

Other menu actions related to folding of XML elements are available from the contextual menu of the folding stripe of the current editor:

- **Close Other Folds (Ctrl+NumPad/ (Command+NumPad/ on OS X))** - Folds all the elements except the current element.
- **Collapse Child Folds (Ctrl+Decimal)** - Folds the elements indented with one level inside the current element.
- **Expand Child Folds (Ctrl+Equals)** - Unfolds all child elements of the currently selected element.
- **Expand All (Ctrl+NumPad+*)** - Unfolds all elements in the current document.
- **Toggle Fold** - Toggles the state of the current fold.

To watch our video demonstration about the folding support in Oxygen XML Editor, go to
<http://oxygentools.com/demo/FoldingSupport.html>.

Outline View

The Outline view offers the following functionality:

- [XML Document Overview](#) on page 226
- [Outline Specific Actions](#) on page 227
- [Modification Follow-up](#) on page 227
- [Document Structure Change](#) on page 227
- [Document Tag Selection](#) on page 228

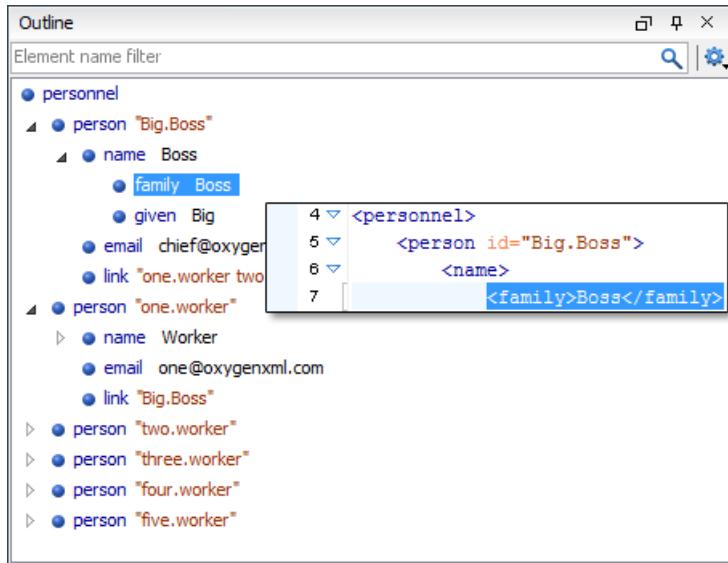


Figure 97: The Outline View

XML Document Overview

The **Outline** view displays a general tag overview of the current edited XML document. It also shows the correct hierarchical dependencies between the tag elements. This functionality makes it easier for the user to be aware of the document structure and the way tags are nested.

The **Outline** view allows you to:

- Insert or delete nodes using pop-up menu actions.
- Move elements by dragging them to a new position in the tree structure.
- Highlight elements in the **Author** editor area.



Note: The **Outline** view is synchronized with the **Author** editor area. When you make a selection in the **Author** editor area, the corresponding elements of the selection are highlighted in the **Outline** view and vice versa. This functionality is available both for single and multiple selection. To deselect one of the elements, use [Ctrl Click \(Command Click on OS X\)](#).

Document errors (such as an element inserted in an invalid position, or a wrong attribute name, or a missing required attribute value) are highlighted in the **Outline** tree:

- A red exclamation mark decorates the element icon.
- A dotted red underline decorates the element name and value.
- A tooltip provides more information about the nature of the error, when you hover with the mouse pointer over the faulted element.

Outline Specific Actions

The following actions are available in the **Settings** menu of the Outline view:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches.

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).

Configure displayed attributes

Displays the [XML Structured Outline preferences page](#).

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Modification Follow-up

When you edit a document, the **Outline** view dynamically follows the changes that you make, displaying the node that you modify in the middle of the view. This functionality gives you great insight on the location of your modifications in the document that you edit.

Document Structure Change

Entire XML elements can be moved or copied in the edited document using only the mouse in the **Outline** view in drag-and-drop operations. Several drag and drop actions are possible:

- If you drag an XML element in the **Outline** view and drop it on another one in the same panel then the dragged element will be moved after the drop target element.
- If you hold the mouse pointer over the drop target for a short time before the drop then the drop target element will be expanded first and the dragged element will be moved inside the drop target element after its opening tag.
- You can also drop an element before or after another element if you hold the mouse pointer towards the upper or lower part of the targeted element. A marker will indicate whether the drop will be performed before or after the target element.
- If you hold down the [**\(Ctrl \(Command on OS X\)\)**](#) key after dragging, there will be performed a copy operation instead of a move one.

The drag and drop action in the **Outline** view can be [disabled and enabled from the Preferences dialog](#).

 **Tip:** You can select and drag multiple nodes in the Author Outline tree.

The Contextual Menu of the Outline View

The following actions are available from the contextual menu in the **Outline** view:

Edit Attributes...

Allows you to edit all the attributes of a selected node. You can find more details about this action in the [Attributes View](#) on page 121 topic.

Edit Profiling Attributes...

Allows you to change the *profiling attributes* defined on all selected elements.

The **Append Child...**, **Insert Before...**, and **Insert After...** actions allow you to quickly insert new tags into the document at the location of the currently selected element. When you select any of these three actions, a content completion window is invoked that offers a list of elements that can be inserted.

Append Child...

Invokes a content completion list with the names of all the elements that are allowed by the associated schema and inserts your selection as a child of the current element.

Insert Before...

Invokes a content completion list with the names of all the elements that are allowed by the associated schema and inserts your selection immediately before the current element, as a sibling.

Insert After...

Invokes a content completion list with the names of all the elements that are allowed by the associated schema and inserts your selection immediately after the current element, as a sibling.

The **Cut**, **Copy**, and **Paste** actions are *the same actions as the Edit menu actions with the same name*, for the currently selected elements.

Toggle Comment

Encloses the currently selected element in an XML comment, if the element is not already commented. If it is already commented, this action will remove the comment.

Rename Element

Invokes a **Rename** dialog that allows you to rename the currently selected element, siblings with the same name, or all elements with the same name.

Expand More

Expands the structure tree of the currently selected element.

Collapse All

Collapses all of the structure tree of the currently selected node.

Document Tag Selection

The Outline view can also be used to search for a specific tag's location and contents in the edited document. Intuitively, by selecting with the left mouse button the desired tag in the Outline view, the document is scrolled to the position of the selected tag. Moreover, the tag's contents are selected in the document, making it easy to notice the part of the document contained by that specific tag and furthermore to easily copy and paste the tag's contents in other parts of the document or in other documents.

You can double click the tag in the **Outline** tree to move focus to the editor.

You can also use key search to look for a particular tag name in the **Outline** tree.

Navigation Buttons

These buttons are available in the editor's main toolbar:

-  **Go to Last Modification** - Moves the cursor to the last modification in any opened document.
-  **Back** - Moves the cursor to the previous position.
-  **Forward** - Moves the cursor to the next position. Enabled after at least one press of the **Back** button.

Using the Go To Dialog

To navigate precisely to a part of the document you are editing in the **Text** mode, use the **Go to** dialog. To open the **Go to** dialog, go to **Find > Go to ... (Ctrl+L (Cmd+L on Mac))**.

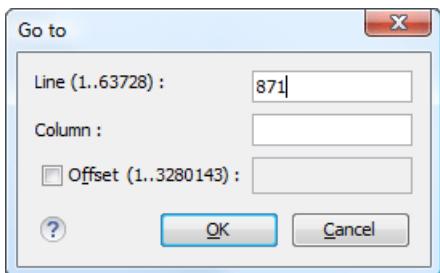


Figure 98: Go to Dialog

Complete the dialog as follows:

- **Line** - destination line in the current document;
- **Column** - destination column in the current document;
- **Offset** - destination offset relative to the beginning of document.

Large Documents

Let's consider the case of documenting a large project. It is likely to be several people involved. The resulting document can be few megabytes in size. How to deal with this amount of data in such a way the work parallelism would not be affected ?

Fortunately, XML provides two solutions for this: DTD entities and XInclude. It can be created a master document, with references to the other document parts, containing the document sections. The users can edit individually the sections, then apply an XSLT stylesheet over the master and obtain the result files, let say PDF or HTML.

Including Document Parts with DTD Entities

There are two conditions for including a part using DTD entities:

- The master document should declare the DTD to be used, while the external entities should declare the XML sections to be referenced.
- The document containing the section must not define again the DTD.

A master document looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "../xml/docbookx.dtd" [
<!ENTITY testing SYSTEM "testing.xml" >]
>
<book>
<chapter> ...
```

The referenced document looks like this:

```
<section> ... here comes the section content ... </section>
```



Note:

The indicated DTD and the element names (*section*, *chapter*) are used here only for illustrating the inclusion mechanism. You can use any DTD and element names you need.

At a certain point in the master document there can be inserted the section *testing.xml* entity:

```
... &testing; ...
```

When splitting a large document and including the separate parts in the master file using external entities, only the master file will contain the Document Type Definition (the DTD) or other type of schema. The included sections can't define

again the schema because the main document will not be valid. If you want to validate the parts separately you have to [use XInclude](#) for assembling the parts together with the master file.

Including Document Parts with XInclude

XInclude is a standard for assembling XML instances into another XML document through inclusion. It enables larger documents to be dynamically created from smaller XML documents without having to physically duplicate the content of the smaller files in the main file. XInclude is targeted as the replacement for External Entities. The advantage of using XInclude is that, unlike the entities method, each of the assembled documents is permitted to contain a Document Type Declaration (DOCTYPE). This means that each file is a valid XML instance and can be independently validated. It also means that the main document to which smaller instances are included can be validated without having to remove or comment out the DOCTYPE, as is the case with External Entities. This makes XInclude a more convenient and effective method for managing XML instances that need to be stand-alone documents and part of a much larger project.

The main application for XInclude is in the document-oriented content frameworks such as manuals and Web pages. Employing XInclude enables authors and content managers to manage content in a modular fashion that is akin to Object Oriented methods used in languages such as Java, C++ or C#.

The advantages of modular documentation include: reusable content units, smaller file units that are easier to be edited, better version control and distributed authoring.

Include a chapter in an article using XInclude

Create a chapter file and an article file in the samples folder of the Oxygen XML Editor install folder.

Chapter file (`introduction.xml`) looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd">
<chapter>
    <title>Getting started</title>
    <section>
        <title>Section title</title>
        <para>Para text</para>
    </section>
</chapter>
```

Main article file looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.docbook.org/xml/4.3/docbookx.dtd"
[ <!ENTITY % xiinclude SYSTEM "../frameworks/docbook/dtd/xinclude.mod">
%xiinclude;
]>
<article>
    <title>Install guide</title>
    <para>This is the install guide.</para>
    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
        href="introduction.dita">
        <xi:fallback>
            <para>
                <emphasis>FIXME: MISSING XINCLUDE CONTENT</emphasis>
            </para>
        </xi:fallback>
    </xi:include>
</article>
```

In this example the following is of note:

- The DOCTYPE declaration defines an entity that references a file containing the information to add the `xi` namespace to certain elements defined by the DocBook DTD.
- The `href` attribute of the `xi:include` element specifies that the `introduction.xml` file will replace the `xi:include` element when the document is parsed.
- If the `introduction.xml` file cannot be found, the parser will use the value of the `xi:fallback` element - a `FIXME` message.

If you want to include only a fragment of a file in the master file, the fragment must be contained in a tag having an `xml:id` attribute and you must use an XPointer expression pointing to the `xml:id` value. For example if the master file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="test.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<test>
  <xi:include href="a.xml" xpointer="a1"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
</test>
```

and the `a.xml` file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<test>
  <a xml:id="a1">test</a>
</test>
```

after resolving the XPointer reference the document is:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="test.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<test>
  <a xml:id="a1" xml:base="a.xml">test</a>
</test>
```

The XInclude support in Oxygen XML Editor is enabled by default. To [toggle it, open the Preferences dialog box](#) and go to **XML > XML Parser > Enable XInclude processing**. When enabled, Oxygen XML Editor will be able to validate and transform documents comprised of parts added using XInclude.

Working with XML Catalogs

An *XML Catalog* maps a system ID or an URI reference pointing to a resource (stored either remotely or locally) to a local copy of the same resource. If XML processing relies on external resources (like referenced schemas and stylesheets, for example), the use of an XML Catalog becomes a necessity when Internet access is not available or the Internet connection is slow.

Oxygen XML Editor supports any XML Catalog file that conforms to one of:

1. [OASIS XML Catalogs Committee Specification v1.1](#)
2. [OASIS Technical Resolution 9401:1997](#) including the plain-text flavor described in that resolution

The version 1.1 of the OASIS XML Catalog specification introduces the possibility to map a system ID, a public ID or a URI to a local copy using only a suffix of the ID or URI used in the actual document. This is done using the catalog elements `systemSuffix` and `uriSuffix`.

Depending on the resource type, Oxygen XML Editor uses different catalog mappings.

Table 7: Catalog Mappings

Document	Referenced Resource	Mappings
XML	DTD	<p><i>system</i> or <i>public</i></p> <p>The Prefer option controls which one of the mappings should be used.</p>
	XML Schema	The following strategy is used (if one step fails to provide a resource, the next is applied): <ol style="list-style-type: none"> 1. resolve the schema using <i>URI</i> catalog mappings. 2. resolve the schema using <i>system</i> catalog mappings.
	Relax NG	
	Schematron	
	NVDL	<p>This happens only if the Resolve schema locations also through system mappings option is enabled (it is by default).</p> <ol style="list-style-type: none"> 3. resolve the root <i>namespace</i> using <i>URI</i> catalog mappings.

Document	Referenced Resource	Mappings
XSL	XSL/ANY	<i>URI</i>
CSS	CSS	<i>URI</i>
XML Schema Schema		The following strategy is used (if one step fails to provide a resource, the next is applied): <ol style="list-style-type: none"> 1. resolve schema reference using <i>URI</i> catalog mappings. 2. resolve schema reference using <i>system</i> catalog mappings. This happens only if the Resolve schema locations also through system mappings option is enabled (it is by default).
Relax NG		<ol style="list-style-type: none"> 3. resolve schema <i>namespace</i> using <i>uri</i> catalog mappings. This happens only if the Process namespaces through URI mappings for XML Schema option is enabled (it is not by default).

An XML Catalog file can be created quickly in Oxygen XML Editor starting from the two XML Catalog document templates called *OASIS XML Catalog 1.0* and *OASIS XML Catalog 1.1* and available in [the document templates dialog](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog
  PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
  "http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd">

<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">

  <!-- Use "system" and "public" mappings when resolving DTDS -->
  <system
    systemId="http://www.docbook.org/xml/4.4/docbookx.dtd"
    uri="frameworks/docbook/4.4/dtd/docbookx.dtd"/>
  <!-- The "systemSuffix" matches any system ID ending in a specified string -->
  <systemSuffix
    systemIdSuffix="docbookx.dtd"
    uri="frameworks/docbook/dtd/docbookx.dtd"/>

  <!-- Use "uri" for resolving XML Schema and XSLT stylesheets -->
  <uri
    name="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng"
    uri="frameworks/docbook/5.0/rng/docbookxi.rng"/>

  <!-- The "uriSuffix" matches any URI ending in a specified string -->
  <uriSuffix
    uriSuffix="docbook.xsl"
    uri="frameworks/docbook/xsl/fo/docbook.xsl"/>

</catalog>
```

Oxygen XML Editor comes with a built-in catalog set as default, but you can also create your own one. Oxygen XML Editor looks for a catalog in the following order:

- user-defined catalog set globally in the [XML Catalog preferences](#) page.
- user-defined catalog set at document type level, in the [Document Type Association preferences pages](#).
- built-in catalogs.

An XML catalog can be used to map a W3C XML Schema specified with an URN in the `xsi:noNamespaceSchemaLocation` attribute of an XML document to a local copy of the schema.

Considering the following XML document code snippet:

```
<topic xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1">
```

The URN can be resolved to a local schema file with a catalog entry like:

```
<uri name="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1"
      uri="topic.xsd"/>
```

Resolve Schemas Through XML Catalogs

Oxygen XML Editor resolves the location of a schema in the following order:

- First, it attempts to resolve the schema location as a URI (`uri`, `uriSuffix`, `rewriteURI` from the XML catalog). If this succeeds, the process ends here.
- If the **Resolve schema locations also through system mappings** option is selected, it attempts to resolve the schema location as a `systemID` (`system`, `systemSuffix`, `rewriteSuffix`, `rewriteSystem` from the XML catalog). If this succeeds, the process ends here.
- If the **Process namespace through URI mappings for XML Schema** option is selected, it attempts to resolve the schema location as a URI (`uri`, `uriSuffix`, `rewriteURI` from the XML catalog). If this succeeds, the process ends here.
- If none of these succeeds, the actual schema location is used.

XML Resource Hierarchy/Dependencies View

The **Resource Hierarchy / Dependencies** view allows you to easily see the hierarchy / dependencies for an XML document. The tree structure presented in this view is built based on the *XInclude* and *External Entity* mechanisms. To define the scope for calculating the dependencies of a resource, click  [Configure dependencies search scope](#) on the **Resource Hierarchy/Dependencies** toolbar.

To open this view, go to **Window > Show View > Resource Hierarchy / Dependencies**. As an alternative, right click the current document and either select **Resource Hierarchy** or **Resource Dependencies**.

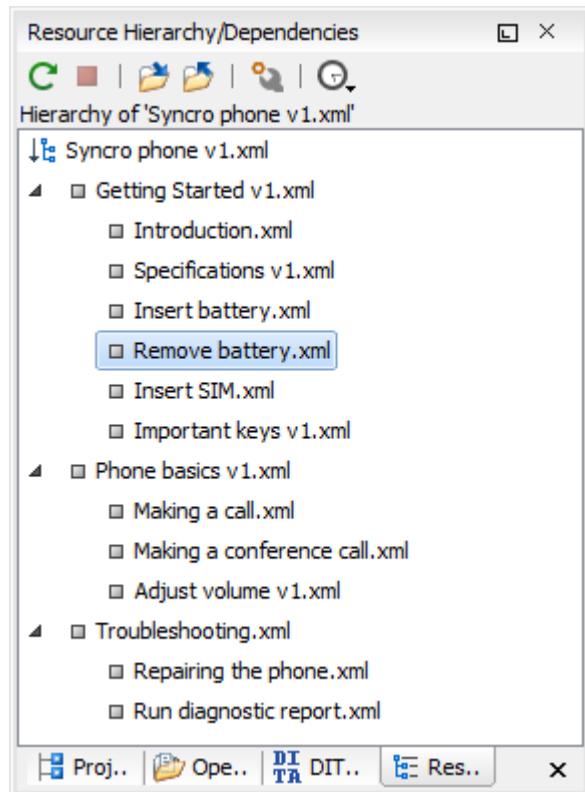


Figure 99: Resource Hierarchy/Dependencies View - Hierarchy for Syncro phone v1.xml

The build process for the dependencies view is started with the **Resource Dependencies** action available on the contextual menu.

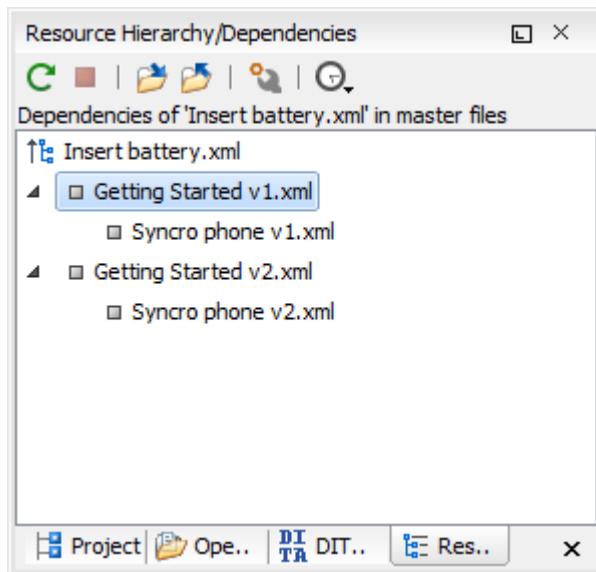


Figure 100: Resource Hierarchy/Dependencies View - Dependencies for Insert battery.xml

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.



Add to Master Files

Adds the currently selected resource in the *Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.



Tip: When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .



Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*. Only the references made through the *XInclude* and *External Entity* mechanisms are handled.

Moving/Renaming XML Resources

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Converting Between Schema Languages

The **Generate/Convert Schema** dialog box allows you to convert a DTD or Relax NG (full or compact syntax) schema or a set of XML files to an equivalent XML Schema, DTD or Relax NG (full or compact syntax) schema. Where perfect equivalence is not possible due to limitations of the target language, Oxygen XML Editor generates an approximation of the source schema. Oxygen XML Editor uses *Trang multi-format converter* to perform the actual schema conversions.

To open the **Generate/Convert Schema** dialog box, select the **Generate/Convert Schema... (Alt Shift C (Command Alt C on OS X))** action from the **Tools** menu or from the **Open with** submenu when invoking the contextual menu in the **Project view**.

A schema being edited can be converted with just one click on a toolbar button if that schema can be the subject of a supported conversion.

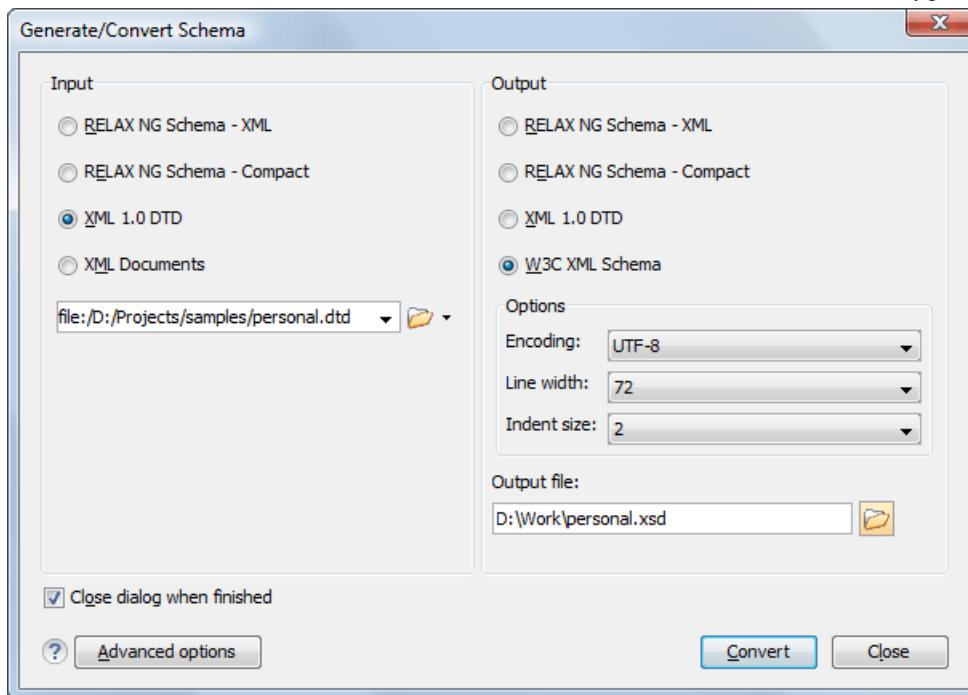


Figure 101: Convert a Schema to Other Schema Language

The language of the source schema is specified with one of the four radio buttons in the **Input** panel. If the conversion is based on a set of XML files, not just a single XML file, select the **XML Documents** option. Then use the file selector to add the XML files involved in the conversion.

The language of the target schema is specified with one of the four options in the **Output** panel. Here you can also choose the encoding, the maximum line width and the number of spaces for one level of indentation.

The conversion can be further fine-tuned by specifying more advanced options available from the **Advanced options** button. For example if the input is a DTD and the output is an XML Schema the following options are available:

Input panel:

- **xmlns** - Specifies the default namespace, that is the namespace used for unqualified element names.
- **xmlns** - Each row specifies the prefix used for a namespace in the input schema.
- **colon-replacement** - Replaces colons in element names with the specified chars when constructing the names of definitions used to represent the element declarations and attribute list declarations in the DTD.
- **element-define** - Specifies how to construct the name of the definition representing an element declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition.
- **inline-atlist** - Instructs the application not to generate definitions for attribute list declarations, but instead move attributes declared in attribute list declarations into the definitions generated for element declarations. This is the default behavior when the output language is XSD.
- **atlist-define** - Specifies how to construct the name of the definition representing an attribute list declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition.
- **any-name** - Specifies the name of the definition generated for the content of elements declared in the DTD as having a content model of ANY.
- **strict-any** - Preserves the exact semantics of ANY content models by using an explicit choice of references to all declared elements. By default, the conversion engine uses a wildcard that allows any element.
- **generate-start** - Specifies whether the conversion engine should generate a start element. DTD's do not indicate what elements are allowed as document elements. The conversion engine assumes that all elements that are defined but never referenced are allowed as document elements.

- **annotation-prefix** - Default values are represented using an annotation attribute `prefix:defaultValue` where prefix is the specified value and is bound to <http://relaxng.org/ns/compatibility/annotations/1.0> as defined by the RELAX NG DTD Compatibility Committee Specification. By default, the conversion engine will use a for prefix unless that conflicts with a prefix used in the DTD.

Output panel:

- **disable-abstract-elements** - Disables the use of abstract elements and substitution groups in the generated XML Schema. This can also be controlled using an annotation attribute.
- **any-process-contents** - One of the values: strict, lax, skip. Specifies the value for the `processContents` attribute of any elements. The default is skip (corresponding to RELAX NG semantics) unless the input format is DTD, in which case the default is strict (corresponding to DTD semantics).
- **any-attribute-process-contents** - Specifies the value for the `processContents` attribute of anyAttribute elements. The default is skip (corresponding to RELAX NG semantics).

Editing XML Tree Nodes

A [Well-Formed XML document](#) can be viewed and edited in Oxygen XML Editor also as a tree of XML elements. This is possible in the Tree Editor perspective, available from **Tools > Tree Editor**. The **Tree Editor** provides specially designed views, toolbars and an editable tree allowing you to execute common tree actions like create/delete nodes, edit node names, move nodes with drag and drop.

If you want to be able to edit XML documents that are not well-formed and still have a tree view of the document you should use the [Outline view](#) in the Editor perspective.

Formatting and Indenting XML Documents

Oxygen XML Editor creates XML documents using several different [edit modes](#). In [text mode](#), you as the author decide how the XML file is formatted and indented. In the other modes, and when you switch between modes, Oxygen XML Editor must decide how to format and indent the XML. Oxygen XML Editor will also format and indent your XML for you in text mode if you use one of the Format and Indent options:

- **Document > Source > Format and Indent** - formats and indents the whole document.
- **Document > Source > Indent Selection** - indents the current selection (but does not add line breaks)
- **Document > Source > Format and Indent Element** - formats and indents the current element (the innermost nested element which contains the current caret) and its child-elements.

A number of settings affect how Oxygen XML Editor formats and indents XML. Many of these settings have to do with how whitespace is handled.

Significant and insignificant whitespace in XML

XML documents are text files that describe complex documents. Some of the white space (spaces, tabs, line feeds, etc.) in the XML document belongs to the document it describes (such as the space between words in a paragraph) and some of it belongs to the XML document (such as a line break between two XML elements). Whitespace belonging to the XML file is called *insignificant whitespace*. The meaning of the XML would be the same if the insignificant whitespace were removed. Whitespace belonging to the document being described is called *significant whitespace*.

Knowing when whitespace is significant or insignificant is not always easy. For instance, a paragraph in an XML document might be laid out like this:

```
<p>
NO Freeman shall be taken or imprisoned, or be disseised of his Freehold, or Liberties, or
free Customs, or be outlawed, or exiled, or any other wise destroyed; nor will We not pass
upon him, nor condemn him, but by lawful judgment of his Peers, or by the Law of the Land.
We will sell to no man, we will not deny or defer to any man either Justice or Right.
</p>
```

By default, XML considers a single whitespace between words to be significant, and all other whitespace to be insignificant. Thus the paragraph above could be written all on one line with no spaces between the start tag and the first word or between the last word and the end tag and the XML parser would see it as exactly the same paragraph. Removing the insignificant space in markup like this is called *normalizing space*.

But in some cases, all the spaces inside an element should be treated as significant. For example, in a code sample:

```
<codeblock>
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}</codeblock>
```

Here every whitespace character between the `codeblock` tags should be treated as significant.

How Oxygen XML Editor determines when whitespace is significant

When Oxygen XML Editor formats and indents an XML document, it introduces or removes insignificant whitespace to produce a layout with reasonable line lengths and elements indented to show their place in the hierarchy of the document. To correctly format and indent the XML source, Oxygen XML Editor needs to know when to treat whitespace as significant and when to treat it as insignificant. However it is not always possible to tell this from the XML source file alone. To determine what whitespace is significant, Oxygen XML Editor assigns each element in the document to one of four categories:

Ignore space

In the ignore space category, all whitespace is considered insignificant. This generally applies to content that consists only of elements nested inside other elements, with no text content.

Normalize space

In the normalize space category, a single whitespace character between character strings is considered significant and all other spaces are considered insignificant. This generally applies to elements that contain text content only. This content can be normalized by removing insignificant whitespace. Insignificant whitespace may then be added to format and indent the content.

Mixed content

In the mixed content category, a single whitespace between text characters is considered significant and all other spaces are considered insignificant. However,

- Whitespace between two child elements embedded in the text is normalized to a single space (rather than to zero spaces as would normally be the case for a text node with only whitespace characters, or the space between elements generally).
- The lack of whitespace between a child element embedded in the text and either adjacent text or another child element is considered significant. That is, no whitespace can be introduced here when formatting and indenting the file.

For example:

```
<p>The file is located in <i>HOME</i>/<i>USER</i>/hello. This is s <strong>big</strong>
<emphasis>deal</emphasis>.
</p>
```

In this example, whitespace should not be introduced around the `i` tags as it would introduce extra significant whitespace into the document. The space between the end `` tag and the beginning `<emphasis>` tag should be normalized to a single space, not zero spaces.

Preserve space

In the preserve space category, all whitespace in the element is regarded as significant. No changes are made to the spaces in elements in this category. Note, however, that child elements may be in a different category, and may be treated differently.

Attribute values are always in the preserve space category. The spaces between attributes in an element tag are always in the default space category.

Oxygen XML Editor consults several pieces of information to assign an element to one of these categories. An element is always assigned to the most restrictive category (from Ignore to Preserve) that it is assigned to by any of the sources Oxygen XML Editor consults. For instance, if the element is named on the **Default elements** list (as described below) but it has an `xml:space="preserve"` attribute in the source file, it will be assigned to the preserve space category. If an element has the `xml:space="default"` attribute in the source, but is listed on the **Mixed content** elements list, it will be assigned to the mixed content category.

To assign elements to these categories, Oxygen XML Editor consults information from the following sources:

xml:space

If the XML element contains the `xml:space` attribute, the element is promoted to the appropriate category based on the value of the attribute.

CSS whitespace property

If the CSS stylesheet controlling the Author mode editor applies the `whitespace: pre` setting to an element, it is promoted to the preserve space category.

CSS display property

If a text node contains only white-spaces:

- If the node has a parent element with the CSS `display` property set to `inline` then the node is promoted to the mixed content category.
- If the left or right sibling is an element with the CSS `display` property set to `inline` then the node is promoted to the mixed content category.
- If one of its ancestors is an element with the CSS `display` property set to `table` then the node is assigned to the ignore space category.

Schema aware formatting

If a schema is available for the XML document, Oxygen XML Editor can use information from the schema to promote the element to the appropriate category. For example:

- If the schema declares an element to be of type `xs:string`, the element will be promoted to the preserve space category because the string built-in type has the whitespace facet with the value `preserve`.
- If the schema declares an element to be mixed content, it will be promoted to the mixed content category.

Schema aware formatting can be turned on and off.

- To turn it on or off for Author mode, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Schema aware > Schema aware normalization, format and indent**.
- To turn it on or off for the Text editing mode, [open the Preferences dialog box](#) and go to **Editor > Format > XML > Schema aware format and indent**.

Preserve space elements list

If an element is listed in the **Preserve space** list in the [XML formatting preferences](#), it is promoted to the preserve space category.

Default space elements list

If an element is listed in the **Default space** list in the [XML formatting preferences](#), it is promoted to the default space category

Mixed content elements list

If an element is listed in the **Mixed content** list in the [XML formatting preferences](#), it is promoted to the mixed content category.

Element content

If an element contains mixed content, that is, a mix of text and other elements, it is promoted to the mixed content category. (Note that, in accordance with these rules, this happens even if the schema declares the element to have element only content.)

If an element contains text content, it is promoted to the default space category.

Text node content

If a text node contains any non-whitespace characters then the text node is promoted to the normalize space category.

An exception to the rule

In general, an element can only be promoted to a more restrictive category (one that treats more whitespace as significant). However, there is one exception. In author mode, if an element is marked as mixed content in the schema, but the actual element contains no text content, it can be demoted to the space ignore category if all of its child elements are displayed as blocks by the associated CSS (that is, they have a CSS property of `display: block`). For example, in some schemas, a section or a table entry can be defined as having mixed content but in many cases they contain only block elements. In these cases, any whitespace they contain cannot be significant and they can be treated as space ignore elements. This exception can be turned on or off using the option [Editor / Edit modes / Author / Schema aware](#).

How Oxygen XML Editor formats and indents XML

You can control how Oxygen XML Editor formats and indents XML documents. This can be particularly important if you store your XML document in a version control system, as it allows you to limit the number of trivial changes in spacing between versions of an XML document. The following settings pages control how XML documents are formatted:

- [Format Preferences](#) on page 1029
- [XML Formatting Preferences](#) on page 1030
- [Whitespaces Preferences](#) on page 1031

When Oxygen XML Editor formats and indents XML

Oxygen XML Editor formats and indents a document, or part of it, on the following occasions:

- In text mode when you select one of the format and indent options (**Document > Source > Format and Indent**, **Document > Source > Indent Selection**, or **Document > Source > Format and Indent Element**).
- When saving documents in Author mode.
- When switching from Author mode to another mode.
- When saving documents in Design mode.
- When switching from Design mode to another mode.
- When saving or switching to Text mode from Grid mode, if the option [Editor / Edit modes / Grid / Format and indent when passing from grid to text or on save](#) is selected.

Setting an Indent Size to Zero

Oxygen XML Editor will automatically [format and indent](#) documents at certain times. This includes indenting the content from the margin to reflect its structure. In some cases you may not want your content indented. To avoid your content being indented, you can set an indent size of zero.

 **Note:** Changing the indent size does not override the rules that Oxygen XML Editor uses for handling whitespace when formatting and indenting XML documents. Indents in elements that require whitespace to be maintained will not have their indent changed by these settings.

There are two cases to consider.

Maintaining zero indent in documents with zero indent

If you have existing documents with zero indent and you want Oxygen XML Editor to maintain a zero indent when editing or formatting those documents:

1. [Open the Preferences dialog box](#) and go to **Editor > Format**.
2. Select **Detect indent on open**.
3. Select **Use zero-indent if detected**.

Oxygen XML Editor will examine the indent of each document as it is opened and if the indent is zero for all lines, or for nearly all lines, a zero indent will be used when formatting and indenting the document. Otherwise, Oxygen XML Editor will use the indent closest to what it detects in the document.

Enforcing zero indent for all documents

If you want all documents to be formatted with zero indent, regardless of their current indenting:

1. [Open the Preferences dialog box](#) and go to **Editor > Format**.
2. Deselect **Detect indent on open**.
3. Set **Indent size** to 0.

All documents will be formatted and indented with an indent of zero.

 **Warning:** Setting the indent size to zero can change the meaning of some file types, such as Python source files.

Format and Indent (Pretty Print) Multiple Files

Oxygen XML Editor provides support for formatting and indenting (*Pretty Print*) multiple files at once. This action is available for any document in XML format, as well as for XQuery, CSS, JavaScript, and JSON documents.

To format and indent multiple files, use the **Format and Indent** action that is available in the contextual menu of the **Project** view. This opens the **Format and Indent** dialog box that allows you to configure options for the action.

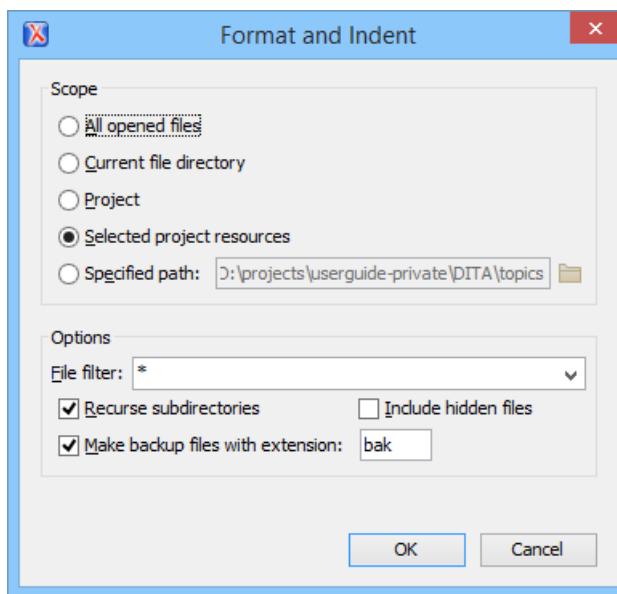


Figure 102: The Format and Indent Dialog Box

The **Scope** section allows you choose from the following scopes:

- **All opened files** - The *pretty print* is performed in all opened files.
- **Directory of the current file** - All the files in the folder of the current edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.

- **Specified path** - *Pretty prints* the files located at a specified path.

The **Options** section includes the following options:

- **File filter** - Allow you to filter the files from the selected scope.
- **Recurse subdirectories** - When enabled, the *pretty print* is performed recursively for the specified scope. The one exception is that this option is ignored if the scope is set to **All opened files**.
- **Include hidden files** - When enabled, the *pretty print* is also performed in the hidden files.
- **Make backup files with extension** - When enabled, Oxygen XML Editor makes backup files of the modified files. The default extension is **.bak**, but you can change the extension as you prefer.

Editing Modular XML Files in the Master Files Context

Smaller interrelated modules that define a complex XML modular structure cannot be correctly edited or validated individually, due to their interdependency with other modules. Oxygen XML Editor provides the support for defining the main module (or modules), allowing you to edit any file from the hierarchy in the context of the master XML files.

You can set a main XML document either using the *master files support from the Project view*, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main XML document. In this case, it considers the current module as the main XML document.

The advantages of editing in the context of a master file include:

- correct validation of a module in the context of a larger XML structure;
- **Content Completion Assistant** displays all collected entities and IDs starting from the master files;
- Oxygen XML Editor uses the schema defined in the master file when you edit a module which is included in the hierarchy through the *External Entity* mechanism;
- the master files defined for the current module determines the *scope of the search and refactoring actions* for ID/IDREFS values and for updating references when renaming/moving a resource. Oxygen XML Editor performs the search and refactoring actions in the context that the master files determine, improving the speed of execution.

To watch our video demonstration about editing modular XML files in the master files context, go to http://oxygenxml.com/demo/Working_With_XML_Modules.html.

Managing ID/IDREFS.

Oxygen XML Editor allows you to search for ID declarations and references (IDREFS) and to *define the scope of the search and refactor operations*. These operations are available for XML documents that have an associated DTD, XML Schema, or Relax NG schema.

Highlight IDs Occurrences in Text Mode

To see the occurrences of an ID in an XML document in the **Text** mode, place the cursor inside the ID declaration or reference. The occurrences are marked in the vertical side bar at the right of the editor. Click a marker on the side bar to navigate to the occurrence that it corresponds to. The occurrences are also highlighted in the editing area.



Note: Highlighted ID declarations are rendered with a different color than highlighted ID references. To customize these colors or disable this feature, *open the Preferences dialog box* and go to **Editor > Mark Occurrences**.

Search and Refactor Actions for ID/IDREFS

Oxygen XML Editor offers full support for managing ID/IDREFS through the search and refactor actions available in the contextual menu. In **Text** mode, these actions are available in the **Quick Assist** menu as well.

The search and refactor actions from the contextual menu are grouped in the **Manage IDs** section:

Rename in

Renames the ID and all its occurrences. Selecting this action opens the **Rename XML ID** dialog. This dialog lets you insert the new ID value and *choose the scope of the rename operation*. For a preview of the changes you are about to make, click **Preview**. This opens the **Preview** dialog, which presents a list with the files that contain changes and a preview zone of these changes.

Rename in File

Renames the ID you are editing and all its occurrences from the current file.

 **Note:** Available in the **Text** mode only.

Search References in

Searches for the references of the ID. Selecting this action opens the *Select the scope for the Search and Refactor operations*.

Search References

Searches for the references of the ID. By default, the scope of this action is the current project. In case you configure a scope using the *Select the scope for the Search and Refactor operations* dialog, this scope will be used instead.

Search Declarations in

Searches for the declaration of the ID reference. Selecting this action opens the *Select the scope for the Search and Refactor operations*.

Search Declarations

Searches for the declaration of the ID reference. By default, the scope of this action is the current project. In case you configure a scope using the *Select the scope for the Search and Refactor operations* dialog, this scope will be used instead.

Search Occurrences in file

Searches for the declaration an references of the ID in the current document.



Note: A quick way to navigate to the declaration of an ID in **Text** mode is to move the cursor over an ID reference and use the Ctrl Click (Command Click on OS X) navigation.

Selecting an ID for which you execute search or refactor operations differs from the **Text** mode to the **Author** mode. In the **Text** mode you position the caret inside the declaration or reference of an ID. In the **Author** mode Oxygen XML Editor collects all the IDs by analyzing each element from the path to the root. In case more IDs are available, you are prompted to choose one of them.

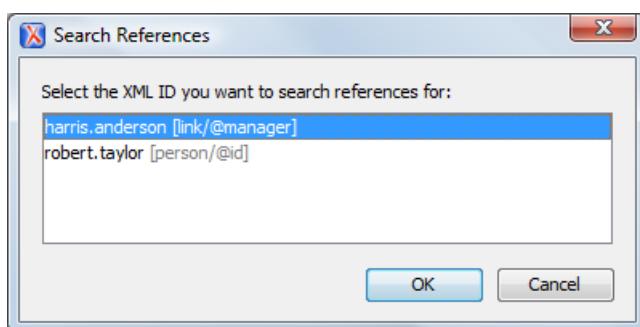


Figure 103: Selecting an ID in the Author Mode

Quick Assist Support for ID/IDREFS in Text Mode

The Quick Assist support is activated automatically when you place the caret inside and ID or an IDREF. To access it, click the yellow bulb help marker placed on the caret line, in the line number stripe of the editor. You can also invoke the quick assist menu if you press **Alt + 1**(**Meta + Alt + 1** on Mac OS X) on your keyboard.

The following actions are available:

Rename in

Renames the ID and all its occurrences. Selecting this action opens the **Rename XML ID** dialog. This dialog lets you insert the new ID value and *choose the scope of the rename operation*. For a preview of the changes you are about to make, click **Preview**. This opens the **Preview** dialog, which presents a list with the files that contain changes and a preview zone of these changes.

Search Declarations

Searches for the declaration of the ID reference. By default, the scope of this action is the current project. In case you configure a scope using the *Select the scope for the Search and Refactor operations* dialog, this scope will be used instead.

Search References

Searches for the references of the ID. By default, the scope of this action is the current project. In case you configure a scope using the *Select the scope for the Search and Refactor operations* dialog, this scope will be used instead.

Change scope

Opens the *Select the scope for the Search and Refactor operations* dialog;

Rename in File

Renames the ID you are editing and all its occurrences from the current file.



Note: Available in the **Text** mode only.

Search Occurrences

Searches for the declaration and references of the ID located at the caret position in the current document.

Search and Refactor Operations Scope

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the **Change scope** operation, available in the Quick Assist action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.

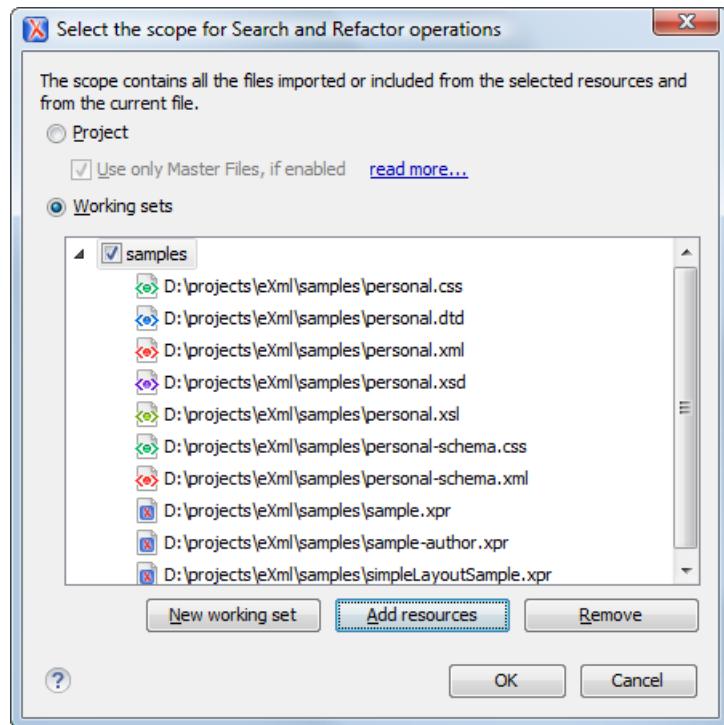


Figure 104: Change Scope Dialog

The scope you define is applied to all future search and refactor operations until you modify it. Contextual menu actions allow you to add or delete files, folders, and other resources to the working set structure.

Viewing Status Information

Status information generated by the **Schema Detection**, **Validation**, **Automatic validation**, and **Transformation** threads are fed into the **Information** view allowing you to monitor how the operation is being executed.

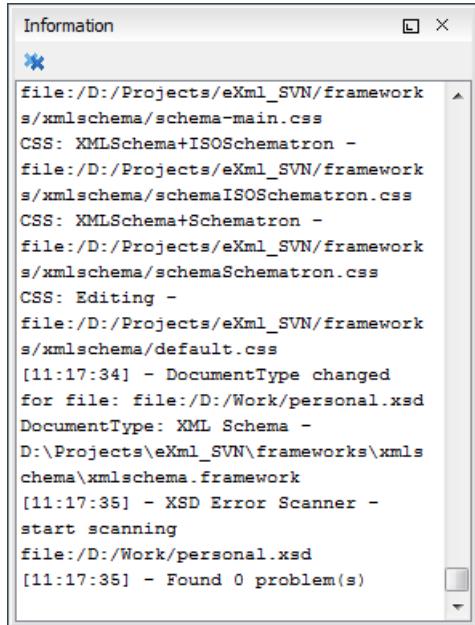


Figure 105: Information view messages

Messages contain a timestamp, the name of the thread that generated it and the actual status information. The number of displayed messages can be controlled from the [Options panel](#).

In order to make the view visible go to menu **Window > Show View > Information**.

Image Preview

Images and SVG files from the **Project** view can be previewed in a separate panel.

To preview an image, either double click the image name or click the **Preview** action from the **Project** view's contextual menu. Supported image types are GIF, JPEG/JPG, PNG, BMP. Once the image is displayed in the **Preview** panel using the actions from the contextual menu, you can scale the image to its original size (1:1 action) or scale it down to fit in the view's available area (**Scale to fit** action).

To preview an [SVG file](#), click the **Preview** action from the **Project** view's contextual menu. Once the SVG is displayed in the **Preview** panel, the following actions are available on the contextual menu: **Zoom in**, **Zoom out**, **Rotate** and **Refresh**.

 **Note:** You can drag an image from the **Image Preview** view and drop it in a DITA, DocBook, or TEI document.

Making a Persistent Copy of Results

The **Results** panel displays the results from the following operations:

- [document validation](#)
- [checking the form of documents](#)
- [XSLT or FO transformation](#)
- [find all occurrences of a string in a file](#)
- [find all occurrences of a string in multiple files](#)

- [applying an XPath expression to the current document](#)

To make a persistent copy of the **Results** panel, use one of these actions:

File > Save Results

Displays the **Save Results** dialog box, used to save the result list of the current message tab. The action is also available on the right click menu of the **Results** panel.

File > Print Results

Displays the **Page Setup** dialog box used to define the page size and orientation properties for printing the result list of the current **Results panel**. The action is also available on the right click menu of the **Results** panel.

Save Results as XML from the contextual menu

Saves the content of the **Results** panel in an XML file with the format:

```
<Report>
  <Incident>
    <engine>The engine who provide the error.<engine>
    <severity>The severity level<severity>
    <Description>Description of output message.</Description>
    <SystemID>The location of the file linked to the message.</SystemID>
    <Location>
      <start>
        <line>Start line number in file.<line>
        <column>Start column number in file<column>
      </start>
      <end>
        <line>End line number in file.<line>
        <column>End column number in file<column>
      </end>
    </Location>
  </Incident>
</Report>
```

Locking and Unlocking XML Markup

For documents with fixed markup such as forms in which the XML tags are not allowed to be modified but only their text content, editing of the XML tag names can be disabled and re-enabled with the **Lock / Unlock the XML tags** action available from:

- The **Document > Source** menu.
- The **Source** submenu from the contextual menu.

You can set the default lock state for all opened editors in the [Preferences XML Editor Format](#) preferences page.

Adjusting the Transparency of XML Markup

Most of the time you want the content of a document displayed on screen with zero transparency. However, if you want to focus your attention only on editing text content inside XML markup, Oxygen XML Editor offers the option of reducing the visibility of the markup by increasing their transparency when displayed in **Text** mode. To change the level

of transparency, use the [Tags Transparency Selector] drop-down list that is available from the **Source** toolbar. By default, this drop-down list is not visible. You can add it to the toolbar by using [the Configure Toolbars action](#). There are several levels of transparency that can be adjusted to make the content more or less visible:

- **Normal contrast** - Resets the transparency level back to normal.
- **Semi-transparent Text** - Slightly reduces the visibility of text to place greater emphasis on the visibility of the XML markup.
- **Transparent Text** - Greatly reduces the visibility of text to place even greater emphasis on the visibility of the XML markup.
- **Semi-transparent Markup** - Slightly reduces the visibility of the XML markup to place greater emphasis on the visibility of the text.

Note: On older versions of Windows (for example, XP or Vista), depending on antialiasing settings and JVM used, this functionality may have no effect.

XML Editor Specific Actions

Oxygen XML Editor offers groups of actions for working on single XML elements. They are available from the **Document** menu and the context menu of the main editor panel.

Split Actions

The editing area can be divided vertically and horizontally by using the following actions that are available in the **Editor** toolbar and **Window** menu:

Split Editor Horizontally

The currently edited file is displayed in two side-by-side editors. Useful when working with documents that require frequent scrolling between two area of interest.

Split Editor Vertically

The currently edited file is displayed in two stacked editors. Useful when working with documents that require frequent scrolling between two area of interest.

Unsplit Editor

Reverts the split editing mode to the usual, single editor, mode.

Edit Actions

The following XML specific editing actions are available in Text mode:

- **Document > Edit > Toggle Line Wrap > Ctrl Shift Y (Command Shift Y on OS X)** Turns on line wrapping in the editor panel if it was off and vice versa. It has the same effect as the [Line wrap](#) preference.
- **Document > Edit > Toggle comment** - Comments the current selection of the current editor. If the selection already contains a comment the action removes the comment from around the selection. If there is no selection in the current editor and the cursor is not positioned inside a comment the current line is commented. If the cursor is positioned inside a comment then the commented text is uncommented. The action is also available on the popup menu of the editor panel.

Select Actions

In Text mode of the XML editor these actions are enabled when the caret is positioned inside a tag name:

- **Document > Select > Element** - Selects the entire current element;
- **Document > Select > Content** - Selects the content of the current element, excluding the start tag and end tag. If it is applied repeatedly, starts with selecting the XML element from the cursor position and extends the selection to the ancestor XML elements. Each execution of the action extends the current selection to the surrounding element;
- **Document > Select > Attributes** - Selects all the attributes of the current element;
- **Document > Select > Parent** - Selects the parent element of the current element;
- Triple click an element or processing instruction - If the triple click is done before the start tag of an element or after the end tag of an element then all the element is selected by the triple click action. If it is done after the start tag or before the end tag then only the element content without the start tag and end tag is selected;
- Triple click an attribute in **Text** mode - If the triple click is performed before the start tag of an attribute or after its end tag, the entire attribute is selected by the triple click action. If it is performed after the start tag or before the end tag, only the attribute content (without the start tag and end tag) is selected;
- Double click after the opening quote or before the closing quote of an attribute value - Select the whole attribute value.

Source Actions

The following actions are available from the **Document > Source** menu or the **Source** submenu when invoking the contextual menu in **Text** mode:

To Upper Case

Converts the content selection to upper case characters.

To Lower Case

Converts the content selection to lower case characters.

Capitalize Lines

It capitalizes the first letter found on every new line that is selected. Only the first letter is affected, the rest of the line remains the same. If the first character on the new line is not a letter then no changes are made.

Join and Normalize Lines

For the current selection, this action joins the lines by replacing the *line separator* with a single space character. It also normalizes the whitespaces by replacing a sequence of such characters with a single space.

Shift Right Tab

Shifts the currently selected block to the right.

Shift Left Shift Tab

Shifts the currently selected block to the left.

& Escape Selection ...

Escapes a range of characters by replacing them with the corresponding character entities.

& Unescape Selection ...

Replaces the character entities with the corresponding characters.

Indent selection Ctrl I (Command I on OS X)

Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences*.

Format and Indent Element Ctrl Shift I

Pretty prints the element that surrounds the current caret position.

Insert XInclude

Displays a dialog that allows you to browse and select the content to be included and automatically generates the corresponding XInclude instruction.

 **Note:** In the **Author** mode, this dialog presents a preview of the inserted document as an author page in the **preview** tab and as a text page in the **Source** tab. In the **Text** mode, the **Source** tab is presented.

& Import entities list

Displays a dialog that allows you to select a list of files as sources for external DTD entities. The internal subset of the DOCTYPE declaration of your document will be updated with the chosen entities. For instance, choosing the files chapter1.xml and chapter2.xml inserts the following section in the DOCTYPE:

```
<!ENTITY chapter1 SYSTEM "chapter1.xml">
<!ENTITY chapter2 SYSTEM "chapter2.xml">
```

Format and Indent

Performs a format and indent (pretty print) action on the current document.

Document > Source > Lock / Unlock the XML Tags

Disables or enables editing of XML tags.

The following actions are only available from the **Source** submenu when invoking the contextual menu in **Text** mode:

Document > Source > Insert new line after

This useful action has the same result with moving the caret to the end of the current line and pressing the *ENTER* key.

Canonicalize

Opens the **Canonicalize** dialog that allows you to select a canonicalization algorithm to standardize the format of the document.

Sign

Opens the **Sign** dialog that allows you to configure a digital signature for the document.

Verify Signature

Allows you to specify the location of a file to verify its digital signature.

XML Document Actions

The **Text** mode of the XML editor provides the following document level actions:

-  **Show Definition** [Ctrl Shift ENTER \(Command Shift ENTER on OS X\)](#), available from the contextual menu of the current editor or the **Document > Schema** menu. Moves the cursor to the definition of the current element or attribute in the schema (DTD, XML Schema, Relax NG schema) associated with the edited XML document. In case the current attribute is “type” belonging to the “<http://www.w3.org/2001/XMLSchema-instance>” namespace, the cursor is moved in the XML schema, to the definition of the type referenced in the value of the attribute.
 -  **Note:** Alternatively you can [Ctrl Click \(Command Click on OS X\)](#) on an element or attribute name to invoke the **Show Definition** action.
- **Copy XPath** ([Ctrl Alt . \(Command Alt . on OS X\)](#)), available from the contextual menu of the current editor or from the **Document > XML Document** menu. Copies the XPath expression of the current element or attribute from the current editor to the clipboard.
-  **Go to Matching Tag** ([Ctrl Shift G \(Command Shift G on OS X\)](#)), available from the **Go to** submenu when invoking the contextual menu of the current editor or from the **Document > XML Document** menu. Moves the cursor to the end tag that matches the start tag, or vice versa.
- **Go after Next Tag** ([Ctrl \] \(Command \] on OS X\)](#)), available from the **Go to** submenu when invoking the contextual menu of the current editor or from the **Document > XML Document** menu. Moves the cursor to the end of the next tag.
- **Go after Previous Tag** ([Ctrl \[\(Command \[on OS X\)](#)), available from the **Go to** submenu when invoking the contextual menu of the current editor or from the **Document > XML Document** menu. Moves the cursor to the end of the previous tag.
-  **Associate XSLT/CSS Stylesheet...**, available from the **Document > XML Document** menu. Inserts an `xmlstylesheet` processing instruction at the beginning of the document referencing either an XSLT or a CSS file depending on the user selection. The referenced stylesheet is used for rendering the document when opened in a Web browser. Referencing the XSLT file is also useful for automatic detection of the XSLT stylesheet when there is no scenario associated with the current document.

When associating the CSS stylesheet, the user can also specify a title for it if it is an alternate one. Setting a *Title* for the CSS makes it the author's preferred stylesheet. Selecting the **Alternate** checkbox makes the CSS an alternate stylesheet.

Oxygen XML Editor fully implements the W3C recommendation regarding [Associating Style Sheets with XML documents](#). See also [Specifying external style sheets](#) in HTML documents.

Also, you can use the [Ctrl Click \(Command Click on OS X\)](#) shortcut to open:

- Any absolute URLs (URLs that have a protocol) regardless of their location in the document.
- URI attributes such as: `schemaLocation`, `noNamespaceSchemaLocation`, `href` and others.
- Processing instructions used for associating resources, `xml-models`, `xml-stylesheets`.

Refactoring Actions

When editing an XML document, the following refactoring actions are available in the **Document > Markup** menu and the **Markup** toolbar:

-  **Surround with Tags** ([Ctrl E \(Command E on OS X\)](#)) - Allows you to choose a tag that encloses a selected portion of content. If there is no selection, the start and end tags are inserted at the caret position.
 - If the **Cursor position between tags** option is set, the caret is placed between the start and end tag.

- If the **Cursor position between tags** option is not set, the caret is placed at the end of the start tag, in an insert-attribute position.
-  **Surround with <tag>** (**Ctrl+I**) - Similar to the **Surround with Tags** action, except that it inserts the last tag used.
-  **Rename Element** - The element from the caret position, and any elements with the same name, can be renamed according with the options from the **Rename** dialog box.
-  **Rename Prefix** - The prefix of the element from the caret position, and any elements with the same prefix, can be renamed according with the options from the **Rename** dialog box.
 - If you select the **Rename current element prefix** option, the application will recursively traverse the current element and all its children.



Note: For example, to change the `xmlns:p1="ns1"` association in the current element to `xmlns:p5="ns1"`, if the `xmlns:p1="ns1"` association is applied on the parent element, then Oxygen XML Editor will introduce `xmlns:p5="ns1"` as a new declaration in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated with another namespace in the current element, then the conflict will be displayed in a dialog box. By pressing **OK**, the prefix is modified from `p1` to `p5` without inserting a new declaration.

- If you select the **Rename current prefix in all document** option, the application will apply the change on the entire document.
- To also apply the action inside attribute values, check the **Rename also attribute values that start with the same prefix** checkbox.
-  **Split element** (**Alt Shift D**) - Split the element from the caret position into two identical elements. The caret must be inside the element.
-  **Join elements** (**Alt Shift J**) - Joins the left and right elements relative to the current caret position. The elements must have the same name, attributes, and attributes values.
-  **Delete element tags** (**Alt Shift X**) - Deletes the start and end tag of the current element.

Smart Editing

The following helper actions are available in the XML editor:

- *Closing tag auto-expansion* - If you want to insert content into an auto closing tag like `<tag/>` deleting the `/` character saves some keystrokes by inserting a separate closing tag automatically and placing the cursor between the start and end tags: `<tag></tag>`
- *Auto-rename matching tag* - When you edit the name of the start tag, Oxygen XML Editor will mirror-edit the name of the matching end tag. This feature can be controlled from the [Content Completion option page](#).
- *Auto-breaking the edited line* - The [Hard line wrap option](#) breaks the edited line automatically when its length exceeds the maximum line length [defined for the format and indent operation](#).
- *Indent on Enter* - The [Indent on Enter option](#) indents the new line inserted when Enter is pressed.
- *Smart Enter* - The [Smart Enter option](#) inserts an empty line between the start and end tags. If Enter is pressed between a start and an end tag the action places the cursor in an indented position on the empty line between the lines that contain the start and end tag.
- *Double click* - A double click selects a different region of text of the current document depending on the position of the click in the document:
 - If the click position is inside a start tag or an end tag then the entire element enclosed by that tag is selected.
 - If the click position is immediately after a start tag or immediately before an end tag then the entire content of the element enclosed by that tag is selected, including all the child elements but excluding the start tag and the end tag of the element.
 - Otherwise, the double click selects the entire current line of text.

Syntax Highlight Depending on Namespace Prefix

The [syntax highlight scheme of an XML file type](#) allows the configuration of a color per each type of token which can appear in an XML file. Distinguishing between the XML tag tokens based on the namespace prefix brings additional visual help in editing some XML file types. For example in XSLT stylesheets elements from different namespaces like XSLT, XHTML, XSL:FO or XForms are inserted in the same document and the editor panel can become cluttered.

[Marking tags with different colors based on the namespace prefix](#) allows easier identification of the tags.

```

3 <xsl:template match="name">
4   <fo:list-item>
5     <fo:list-item-label end-indent="label-end0">
6       <fo:block text-align="end" font-weight="bold">Full Name</fo:block>
7     </fo:list-item-label>
8     <fo:list-item-body start-indent="body-start0">
9       <xsl:apply-templates select="*"/>
10      <fo:list-item-body>
11    </fo:list-item>
12 </xsl:template>

```

Figure 106: Example of Coloring XML Tags by Prefix

Editor Highlights

An editor highlight is a text fragment emphasized by a colored background. Highlights are generated in both **Editor** and **Author** mode, when the following actions generate results: **XPath**, **Find All**, **Find in Files**, **Search References**, and **Search Declarations**.

By default, Oxygen XML Editor uses a different color for each type of highlight: *XPath*, *Find*, *Search References*, and *Search Declarations*. You can customize these colors and the maximum number of highlights displayed in a document on the [Editor preferences page](#). The default maximum number of highlights is 10000.

You are able to navigate in the current document through the highlights using one of the following methods:

- Clicking the markers from the range ruler, located at the right side of the document.
- Clicking the **Next** and **Previous** buttons from the bottom of the range ruler.



Note: When there are multiple types of highlights in the document, the **Next** and **Previous** buttons navigate through highlights of the same type.

- Clicking the messages displayed in the [Results view](#).

To remove the highlights, you can:

- Click the **Remove all** button from bottom of the range ruler.
- Close the results tab that contains the output of the action that generated the highlights.
- Click the **Remove all** button from the results panel.



Note: Use the **Highlight all results in editor** button to either display all the highlights or hide them.

Batch Editing Actions on Highlights

Working with XML documents implies frequent changes to structure and content. You are often faced with a situation where you need to make a slight change in hundreds of places in the same document. Oxygen XML Editor introduced a new feature, **Manage Highlighted Content**, designed to help you achieve that.

When you are in **Text** mode and you perform a search operation or apply an XPath that highlights more than one result, you can select the **Manage Highlighted Content** action from the contextual menu of any highlight in the document. In the sub-menu, the following options are available:

- **Modify All** - Use this option to modify in-place all the occurrences of the selected content. When you use this option, a thin rectangle replaces the highlights and lets you start editing;



Note: In case you select a very large number of highlights that you want to modify using this feature, when you select this option, a dialog box informs you that you may experience performance issues. You have the option to either use the **Find/Replace** dialog box, or continue the operation.

- **Surround All** - Use this option to surround the content with a specific tag. This option opens the **Tag** dialog box. The **Specify the tag** drop-down presents all the available elements that you can choose from.
- **Remove All** - Removes all the highlighted content.

In case you right click a different part of the document than a highlight, you only have the option to select **Modify All Matches**.

XML Quick Fixes

The Oxygen XML Editor Quick Fix support helps you resolve errors that appear in an XML document by offering quick fixes to problems such as missing required attributes or invalid elements. This section explains the quick fix support for XSD, Relax NG, and Schematron validation errors.

To activate this feature, place the caret in the highlighted area of text where a validation error occurs. If a Quick Fix is available for that particular error, the icon is displayed in the stripe on the left side of the editor. If you click this icon, Oxygen XML Editor displays the list of available fixes. You can also invoke the quick fix menu by pressing **Alt 1** (**Command Alt 1 on OS X**) on your keyboard.

Whenever you make a modification in the XML document or you apply a fix, the list of quick fixes is recomputed to ensure that you always have valid proposals.



Note: A quick fix that adds an element inserts it along with required and optional elements, and required and fixed attributes, depending on how the *Content Completion Assistant options* are set.

Quick Fixes for XSD and Relax NG Errors

Oxygen XML Editor offers quick fixes for common errors that appear in XML documents. Quick fixes are available for XML documents that are validated against XSD or Relax NG schemas.



Note: For XML documents validated against XSD schemas, the quick fixes are only available if you use the default Xerces validation engine.

Quick fixes are available in **Text** mode and **Author** mode:



Oxygen XML Editor provides quick fixes for numerous problems, including:

Problem type	Available quick fixes
A specific element is required in the current context	Insert the required element
An element is invalid in the current context	Remove the invalid element
The content of the element should be empty	Remove the element content
An element is not allowed to have child elements	Remove all child elements
Text is not allowed in the current element	Remove the text content
A required attribute is missing	Insert the required attribute
An attribute is not allowed to be set for the current element	Remove the attribute

Problem type	Available quick fixes
The attribute value is invalid	Propose the correct attribute values
ID value is already defined	Generate a unique ID value
References to an invalid ID	Change the reference to an already defined ID

Schematron Quick Fixes (SQF)

Oxygen XML Editor provides support for Schematron Quick Fixes (SQF). They help you resolve errors that appear in XML documents that are validated against Schematron schemas by offering you solution proposals. The Schematron Quick Fixes are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. Specifically, they are associated with *assert* or *report* messages.

Displaying the Schematron Quick Fix Proposals

The defined Schematron Quick Fixes are displayed on validation errors in **Text** mode and **Author** mode.

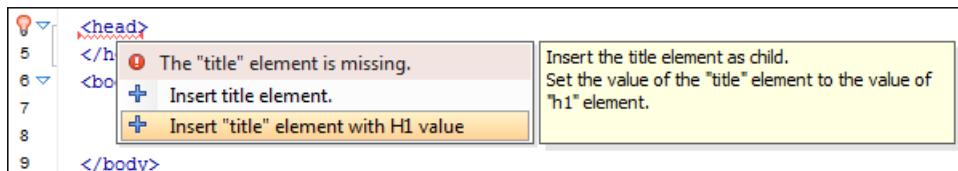


Figure 107: Example of a Schematron Quick Fix

Refactoring XML Documents

In the life cycle of XML documents there are instances when the XML structure needs to be changed to accommodate various needs. For example, when an associated schema is updated, an attribute may have been removed, or a new element added to the structure.

These types of situations cannot be resolved with a traditional *Find/Replace* tool, even if the tool accepts regular expressions. The problem becomes even more complicated if an XML document is computed or referenced from multiple modules, since multiple resources need to be changed.

To assist you with these types of refactoring tasks, Oxygen XML Editor includes a specialized **XML Refactoring** tool that helps you manage the structure of your XML documents.

XML Refactoring Tool

The **XML Refactoring** tool is presented in the form of an easy to use wizard that is designed to reduce the time and effort required to perform various structure management tasks. For example, you can insert, delete, or rename an attribute in all instances of a particular element that is found in all documents within your project.

To access the tool, select the **XML Refactoring...** action from one of the following locations:

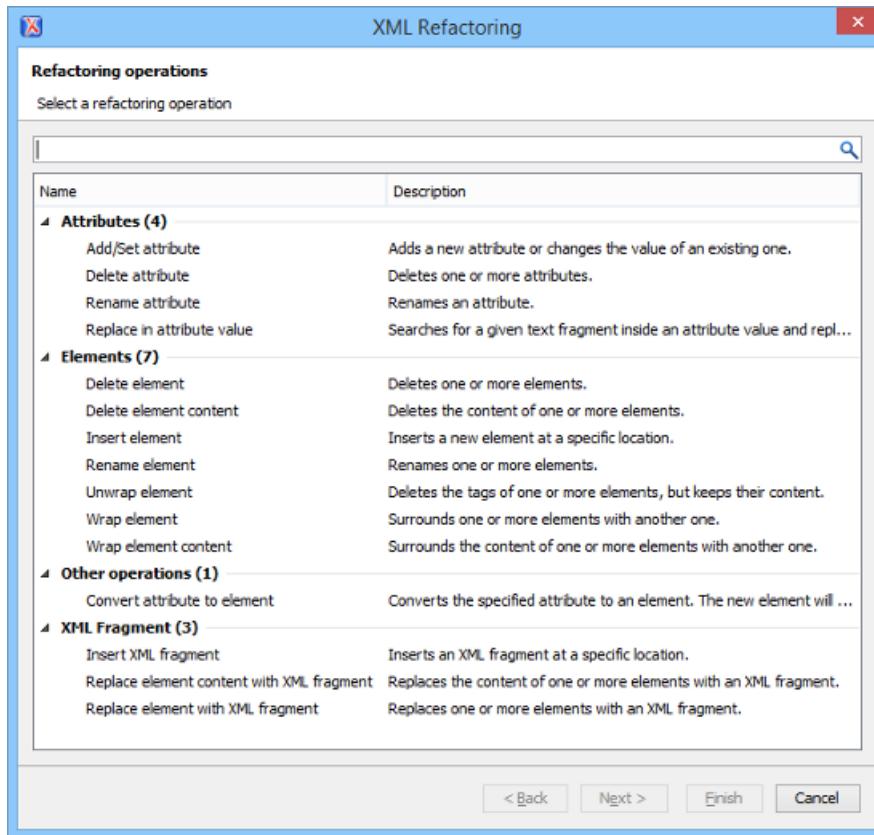
- The **Tools** menu.
- The **Refactoring** submenu from the contextual menu in the **Project** view.
- The **Refactoring** submenu from the contextual menu in the **DITA Maps Manager** view.

The tool includes the following wizard pages:

Refactoring operations

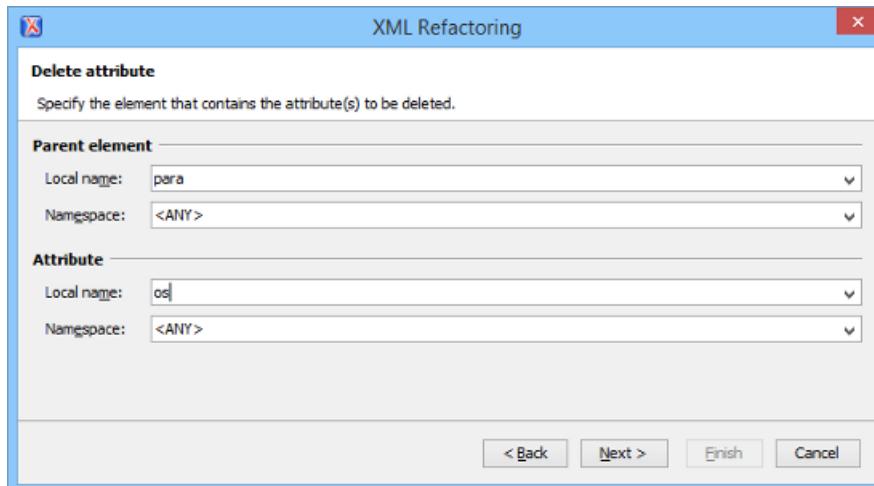
The first wizard page displays, and allows you to select, the available operations, which are grouped by category.

To search for an operation, you can use the filter text box at the top of the page.



Configure Operation Parameters

The next wizard page allows you to specify the parameters for the refactoring operation. The parameters are specific to the type of refactoring operation that is being performed. For example, to delete an attribute you need to specify the parent element and the qualified name of the attribute to be removed.



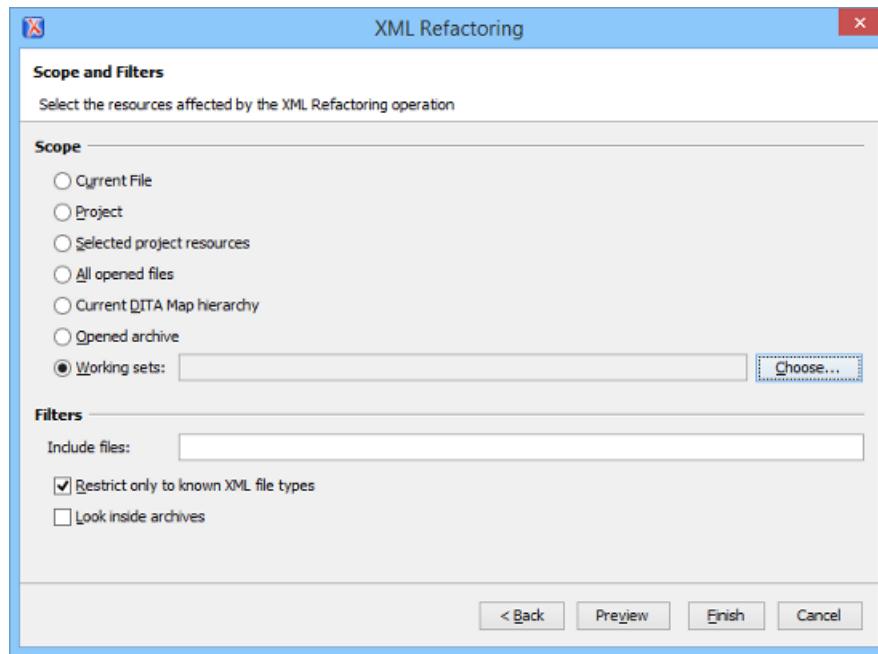
Scope and Filters

The last wizard page allows you to select the set of files that represent the input of the operation. You can select from predefined resource sets (such as the current file, your whole project, the current DITA Map hierarchy, etc.) or you can define your own set of resources by creating a working set.

The **Filters** section includes the following options:

- **Include files** - Allows you to filter the selected resources by using a file pattern. For example, to restrict the operation to only analyze build files you could use build*.xml for the file pattern.

- **Restrict only to known XML file types** - When enabled, only resources with a known XML file type will be affected by the operation.
- **Look inside archives** - When enabled, the resources inside archives will also be affected.



If an operation takes longer than expected you can use the **Stop** button in the progress bar to cancel the operation.



Note: It is recommended that you use the **Preview** button to review all the changes that will be made by the refactoring operation before applying the changes.



Warning: After clicking the **Finish** button, the operation will be processed and Oxygen XML Editor provides no automatic means for reverting the operations. Any **Undo** action will only revert changes on the current document.

Predefined Refactoring Operations

The XML Refactoring tool includes a variety of predefined operations that can be used for common refactoring tasks. They are grouped by category in the **Refactoring operations** wizard page and include the following operations:

Refactoring Operations for Attributes

1. **Add/Change attribute** - Use this operation to change the value of an attribute or insert a new one. To perform this operation, specify the following parameters:
 - The **Local name** and **Namespace** for the *Parent element*.
 - The **Local name**, **Namespace**, and **Value** of the affected *Attribute*.
 - One of the following choices for the **Operation mode** in the *Options* section:
 - **Add the attribute in the parent elements where it is missing**
 - **Change the value in the parent elements where the attribute already exists**
 - **Both**
2. **Delete attribute** - Use this operation to remove one or more attributes. To perform this operation, specify the following parameters:
 - The **Local name** and **Namespace** for the *Parent element*.
 - The **Local name** and **Namespace** for the *Attribute* to be removed.

3. **Rename attribute** - Use this operation to rename an attribute. Specify the following parameters in the **Rename attribute** dialog box:
 - The **Local name** and **Namespace** for the *Parent element*.
 - The **Local name**, **Namespace**, and **New local name** of the *Attribute*.
4. **Replace in attribute value** - Use this operation to search for a text fragment inside an attribute value and change the fragment to a new value. To perform this operation, specify the following parameters:
 - The **Local name** and **Namespace** for the *Parent element*.
 - The **Local name** and **Namespace** for the *Attribute* to be modified.
 - The text *Fragments to Find* and **Replace with**.



Note: You can use Perl-like regular expressions when specifying the text to find. The **Replace with** parameter can bind regular expression capturing groups (\$1, \$2, etc.) from the find pattern.

Refactoring Operations for Elements

1. **Delete element** - Use this operation to delete elements. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression.
2. **Delete element content** - Use this operation to delete content of elements. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression.
3. **Insert element** - Use this operation to insert new element. To perform this operation, specify the following parameters:
 - The **Local name** and **Namespace** for the *element* to be inserted.
 - The *Location* of the new element in the form of an **XPath** expression and its **Position**. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes. The possible selections in the **Position** drop-down list include **After**, **Before**, **First child**, or **Last child**.
4. **Rename element** - Use this operation to rename elements. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
 - The **New local name** of the element.
5. **Unwrap element** - Use this operation to delete the surrounding tags of elements, while keeping their content unchanged. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
6. **Wrap element** - Use this operation to surround elements with element tags. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
 - The **Local name** and **Namespace** of the *Wrapper element*.
7. **Wrap element content** - Use this operation to surround the content of elements with element tags. To perform this operation, specify the following parameters:

- The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
- The **Local name** and **Namespace** of the *Wrapper element* in which its content will be wrapped.

Refactoring Operations for XML Fragments

- 1. Insert XML fragment** - Use this operation to insert an XML fragment. To perform this operation, specify the following parameters:
 - The **XML Fragment** to be inserted.
 - The *Location* of the new fragment in the form of an **XPath** expression and its **Position**. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes. The possible selections in the **Position** drop-down list include **After**, **Before**, **First child**, or **Last child**.
- 2. Replace element content with XML fragment** - Use this operation to replace the content of elements with an XML fragment. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
 - The **XML Fragment** with which to replace the content of the target elements.
- 3. Replace element with XML fragment** - Use this operation to replace elements with an XML fragment. To perform this operation, specify the following parameters:
 - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
 - The **XML Fragment** with which to replace the target elements.

Additional Notes

-  **Note:** There are some operations that allow <ANY> for the **local name** and **namespace** parameters. This value can be used to select an element or attribute regardless of its local name or namespace. Also, the <NO_NAMESPACE> value can be used to select nodes that do not belong to a namespace.
-  **Note:** Some operations have parameters that accept XPath expressions to match elements or attributes. In these XPath expressions you can only use the prefixes declared in the [Options > Preferences > XML > XSLT-FO-XQUERY > XPath](#) page. This preferences page can be easily opened by clicking on the link in the note (**Each prefix used in an XPath expression must be declared in the Default prefix-namespace mappings section**) at the bottom of the **Configure Operation Parameters** wizard page.

Custom Refactoring Operations

If none of the predefined operations will help you accomplish a particular refactoring task, you can create a custom operation that is specific to your needs. For example, if you want to convert an attribute to an element and insert the element as the first child of the parent element, a custom refactoring operation needs to be created.

-  **Note:** The custom refactoring operations are only available in the Enterprise edition.

An XML Refactoring operation is defined as a pair of resources:

- An XQuery Update script file that Oxygen XML Editor will run in order to refactor the XML files.
- An *XML Operation Descriptor* file that contains information about the operation, such as the name, description, and parameters.

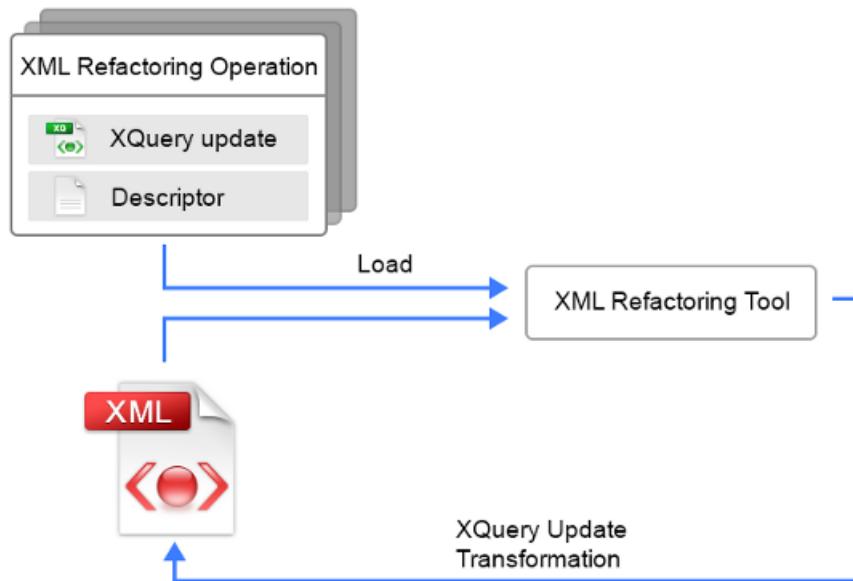


Figure 108: Diagram of an XML Refactoring Operation

All the defined operations are loaded by the XML Refactoring tool and presented in [the Refactoring operations wizard page](#).

After the user chooses an operation and specifies its parameters, an XQuery Update transformation is processed over the input file. This transformation is executed in a *safe mode*, which implies the following:

- When loading the document:
 - The XInclude mechanism is disabled. This means that the resources included by using XInclude will not be visible in the transformation.
 - The DTD entities will be processed without being expanded.
 - The associated DTD will be not loaded, so the default attributes declared in the DTD will not be visible in the transformation.
- When saving the updated XML document:
 - The DOCTYPE will be preserved.
 - The DTD entities will be preserved as they are in the original document when the document is saved.
 - The attribute values will be kept in their original form without being normalized.
 - The spaces between attributes are preserved. Basically, the spaces are lost by an XML serialization since they are not considered important.

The result of this transformation overrides the initial input file.

Note: To achieve some of the previous goals, the XML Refactoring mechanism adds several attributes that are interpreted internally. The attributes belong to the `http://oxygenxml.com/app/xml_refactory/additional_attributes` namespace. These attributes should not be taken into account when processing the input XML document, since they are discarded when the transformed document is serialized.

Restriction: *Comments or processing instructions* that are in any node before or after the root element cannot be modified by an XML Refactoring operation. In other words, XML Refactoring operations can only be performed on *comments or processing instructions* that are inside the root element.

Creating a Custom Refactoring Operation

To create a custom refactoring operation, follow these steps:

1. [Create an XQuery Update script file.](#)
2. [Create an XML Refactoring Operation Descriptor file.](#)
3. Store both files in *one of the locations that Oxygen XML Editor* scans when loading the custom operations.

Once you run the **XML Refactoring** tool again, the custom operation appears in [the Refactoring operations wizard page](#).

Custom Refactoring XQuery Update Script

The first step in creating a custom refactoring operation is to create an XQuery Update script. The easiest way to create this file is to use the **New** document wizard and create a new **XQuery** file.

There are cases when it is necessary to add parameters in the XQuery script. For instance, if you want to rename an element you may want to declare an external parameter associated with the name of the element to rename. To allow you to specify the value for these parameters, they need to be declared in [the refactoring operation descriptor file](#) that is associated with this operation.

 **Note:** The XQuery Update processing is disabled by default in Oxygen XML Editor. Thus, if you want to create or edit an XQuery Update script you have to enable this facility by creating an [XQuery transformation scenario](#) and choose the **Saxon EE** as the transformation engine. Also, you need to make sure the **Enable XQuery update** option is enabled in the [Saxon processor advanced options](#).

The next step in creating a custom refactoring operation is to [create a custom operation descriptor file](#).

Custom Refactoring Operation Descriptor File

The second step in creating a custom refactoring operation is to create an operation descriptor file. The easiest way to do this is to use the **New** document wizard and choose the **XML Refactoring Operation Descriptor** template.

Introduction to the Descriptor File

This file contains information (such as `name`, `description`, and `id`) that is necessary when loading an XML Refactoring operation . It also contains the path to the XQuery Update script that is associated with the particular operation through the `script` element.

You can specify a `category` for your custom operations to logically group certain operations. The `category` element is optional and if it is not included in the descriptor file, the default name of the category for the custom operations is *Other operations*.

The descriptor file is edited and validated against the following schema:
`frameworks/xml_refactoring/operation_descriptor.xsd`.

Declaring Parameters in the Descriptor File

If the XQuery Update script includes parameters, they should be declared in the `parameters` section of the descriptor file. All the parameters specified in this section of the descriptor file will be displayed in the **XML Refactoring** tool within [the Configure Operation Parameters wizard page](#) for that particular operation.

The value of the first `description` element in the `parameters` section will be displayed at the top of [the Configure Operation Parameters wizard page](#).

To declare a parameter, specify the following information:

- **label** - This value is displayed in the user interface for the parameter.
- **name** - The parameter name used in the XQuery Update script and it should be the same as the one declared in the XQuery script.
- **type** - Defines the type of the parameter and how it will be rendered. There are several types available:
 - **TEXT** - Generic type used to specify a simple text fragment.

- XPATH - Type of parameter whose value is an XPATH expression. For this type of parameter, Oxygen XML Editor will use a text input with corresponding content completion and syntax highlighting.



Note: The value of this parameter is transferred as plain text to the XQuery Update transformation without being evaluated. To evaluate it in the XQuery Update script you could use the `saxon:evaluate` Saxon extension function.



Note: A relative XPath expression is converted to an absolute XPath expression by adding `//` before it (`//XPathExp`). This conversion is done before transferring the XPath expression to the XQuery Update engine.



Note: When writing XPath expressions, you can only use prefixes declared in the [Options > Preferences > XML > XSLT-FO-XQUERY > XPath](#) options page.

- NAMESPACE - Used for editing namespace values.
- REG_EXP_FIND - Used when you want to match a certain text by using Perl-like regular expressions.
- REG_EXP_REPLACE - Used along with REG_EXP_FIND to specify the replacement string.
- XML_FRAGMENT - This type is used when you want to specify an XML fragment. For this type, Oxygen XML Editor will display a text area specialized for inserting XML documents.
- NC_NAME - The parameter for NC_NAME values. It is useful when you want to specify the local part of a *QName* for an element or attribute.
- BOOLEAN - Used to edit boolean parameters.
- TEXT_CHOICE - It is useful for parameters whose value should be from a list of possible values. Oxygen XML Editor renders each possible value as a radio button.
- **description** - The description of the parameter. It is used by the application to display a tooltip when you hover over the parameter.
- **possibleValues** - Contains the list with possible values for the parameter and you can specify the default value, as in the following example:

```
<possibleValues onlyPossibleValuesAllowed="true">
  <value name="before">Before</value>
  <value name="after" default="true">After</value>
  <value name="firstChild">First child</value>
  <value name="lastChild">Last child</value>
</possibleValues>
```

Specialized Parameters (elementParameter and attributeParameter)

If you want to match elements or attributes, use the specialized parameters `elementParameter` or `attributeParameter`. If you use these parameters, Oxygen XML Editor will propose all declared elements or attributes based on the schema associated with the currently edited file.

Example of an `elementParameter`:

```
<elementParameter id="elemID">
  <localName label="Name" name="element_localName" allowsAny="true">
    <description>The local name of the attribute's parent element.</description>
  </localName>
  <namespace label="Namespace" name="element_namespace" allowsAny="true">
    <description>The local name of the attribute's parent element</description>
  </namespace>
</elementParameter>
```

This parameter is used to specify elements by local name and namespace. For this type of parameter, the application displays two combo boxes with elements and namespaces collected from the associated schema of the currently edited file. The text from the `label` attribute is displayed in the application as label of the associated combo. The `name` attribute is used to specify the name of the parameter from the XQuery Update script. If you specify the `allowsAny` attribute, the application will propose `<ANY>` as a possible value for the `Name` and `Namespace` combo boxes.

Example of an attributeParameter:

```
<attributeParameter dependsOn="elemID">
  <localName label="Name" name="attribute_localName">
    <description>The name of the attribute to be converted.</description>
  </localName>
  <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
    <description>The namespace of the attribute to be converted.</description>
  </namespace>
</attributeParameter>
```

This parameter is used to specify attributes by local name and namespace. For this type of parameter, the application displays two combo boxes with attributes and their namespaces collected from the associated schema of the currently edited file. The text from the label attribute is displayed in the application as the label of the associated combo box.

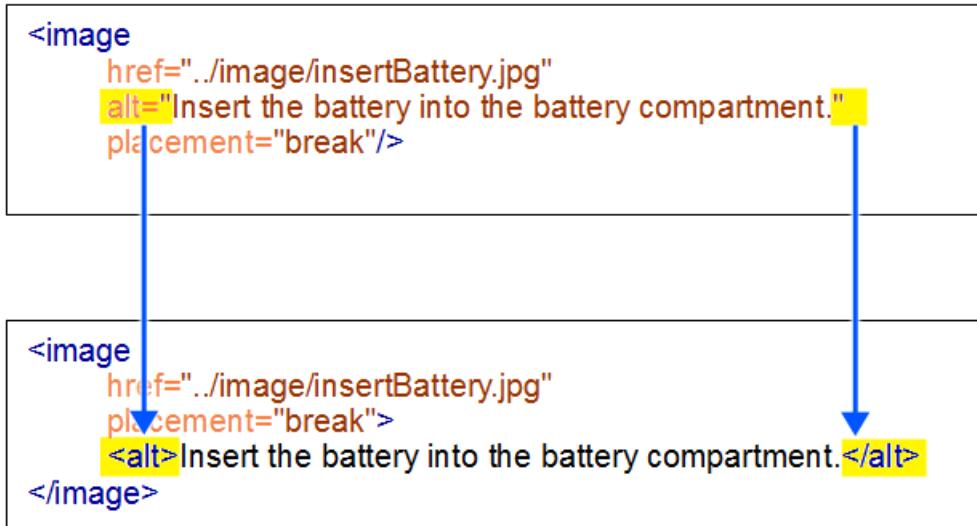
 **Note:** An attributeParameter is dependant upon an elementParameter. The list of attributes and namespaces are computed based on the selection in the elementParameter combo boxes.

 **Note:** All predefined operations are loaded from the installDir/refactoring folder.

Example of an XML Refactoring Operation

To demonstrate creating a custom operation, consider that we have a task where we need to convert an attribute into an element and insert it inside another element. A specific example would be if you have a project with a variety of image elements where a deprecated alt attribute was used for the description and you want to convert all instances of that attribute into an element with the same name and insert it as the first child of the image element.

Thus, our task is to convert this attribute into an element with the same name and insert it as the first child of the image element.



A new operation requires an XQuery Update script and an XML Refactoring operation descriptor file.

Sample XQuery Update Script for Creating a Custom Operation to Convert an Attribute to an Element

The XQuery Update script does the following:

- Iterates over all elements from the document that have the specified local name and namespace.
- Finds the attribute that will be converted to an element.
- Computes the QName of the new element to be inserted and inserts it as the first child of the parent element.

```
(:
  XQuery document used to implement 'Convert attribute to element' operation from XML Refactoring tool.
  :)
```

```

declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method    "xml";
declare option output:indent   "no";

(: Local name of the attribute's parent element. :)
declare variable $element_localName as xs:string external;

(: Namespace of the attribute's parent element. :)
declare variable $element_namespace as xs:string external;

(: The local name of the attribute to be converted :)
declare variable $attribute_localName as xs:string external;

(: The namespace of the attribute to be converted :)
declare variable $attribute_namespace as xs:string external;

(: Local name of the new element. :)
declare variable $new_element_localName as xs:string external;

(: Namespace of the new element. :)
declare variable $new_element_namespace as xs:string external;

(: Convert attribute to element:)
for $node in //*
(: Find the attribute to convert :)
let $attribute :=
  $node/@*[local-name() = $attribute_localName and
  ($attribute_namespace = '<ANY>' or $attribute_namespace = namespace-uri())]

(: Compute the prefix for the new element to insert :)
let $prefix :=
  for $p in in-scope-prefixes($node)
    where $new_element_namespace = namespace-uri-for-prefix($p, $node)
return $p

(: Compute the qname for the new element to insert :)
let $new_element_qName :=
  if (empty($prefix) or $prefix[1] = '') then $new_element_localName
  else $prefix[1] || ':' || $new_element_localName
  where ('<ANY>' = $element_localName or local-name($node) = $element_localName) and
        ($element_namespace = '<ANY>' or $element_namespace = namespace-uri($node))

return
  if (exists($attribute)) then
    (insert node element {QName($new_element_namespace, $new_element_qName)})
    {string($attribute)} as first into $node,
    delete node $attribute)
  else ()

```

Sample Operation Descriptor File for Creating a Custom Operation to Convert an Attribute to an Element

After you have developed the XQuery script, you have to create an XML Refactoring operation descriptor. This descriptor is used by application to load the operation details such as name, description, or parameters.

```

<?xml version="1.0" encoding="UTF-8"?>
<refactoringOperationDescriptor
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://oxygenxml.com/ns/xmlRefactoring"
  id="convert-attribute-to-element"
  name="Convert attribute to element">
  <description>Converts the specified attribute to an element. The new element will be inserted as first child of the attribute's parent element.</description>
  <script type="XQUERY_UPDATE" href="convert-attribute-to-element.xq"/>
  <parameters>
    <description>Specify the attribute to be converted to element.</description>
    <section label="Parent element">
      <elementParameter id="elemID">
        <localName label="Name" name="element_localName" allowsAny="true">
          <description>The local name of the attribute's parent element.</description>
        </localName>
        <namespace label="Namespace" name="element_namespace" allowsAny="true">
          <description>The local name of the attribute's parent element</description>
        </namespace>
      </elementParameter>
    </section>
    <section label="Attribute">
      <attributeParameter dependsOn="elemID">
        <localName label="Name" name="attribute_localName">
          <description>The name of the attribute to be converted.</description>
        </localName>
        <namespace label="Namespace" name="attribute_namespace" allowsAny="true">

```

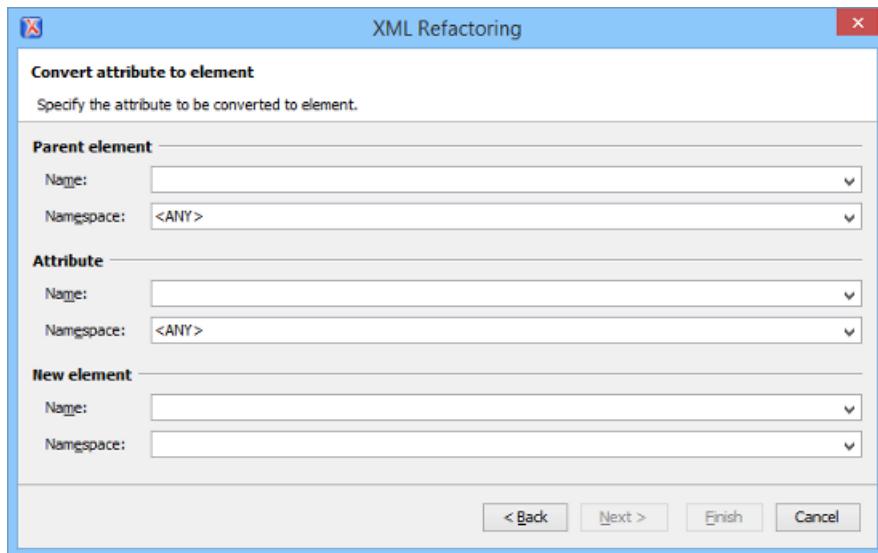
```

<description>The namespace of the attribute to be converted.</description>
</namespace>
</attributeParameter>
</section>
<section label="New element">
  <elementParameter>
    <localName label="Name" name="new_element_localName">
      <description>The name of the new element.</description>
    </localName>
    <namespace label="Namespace" name="new_element_namespace">
      <description>The namespace of the new element.</description>
    </namespace>
  </elementParameter>
</section>
</parameters>
</refactoringOperationDescriptor>

```

After you have created these files, copy them into a folder [scanned by Oxygen XML Editor when it loads the custom operation](#). When the XML Refactoring tool is started again, you will see the created operation.

Since various parameters can be specified, this custom operation can also be used for other similar tasks. The following image shows the parameters that can be specified in our example of the custom operation to convert an attribute to an element:



Storing and Sharing Refactoring Operations

Oxygen XML Editor scans the following locations when looking for XML Refactoring operations to provide flexibility:

- A refactoring folder, created inside a directory that is associated to a *framework* you are customizing.
- Any folder. In this case, you need to [open the Preferences dialog box](#), go to **XML > XML Refactoring**, and specify the same folder in the **Load additional refactoring operations from** text box.

 **Note:** If you share a project with your team, you can also share the custom operation by saving them in a folder that is part of your project. Then switch the **XML Refactoring** option page at project level ([open the Preferences dialog box](#), go to **XML > XML Refactoring**, and select **Project Options** at the bottom of the dialog box), and in the **Load additional refactoring operations from** text box, use the \${pd} editor variable so that the folder path is declared relative to the project.

- A folder specified by the [XML Refactoring Operations Plugin Extension](#).
- The refactoring folder from the Oxygen XML Editor installation directory ([oxygen Installation Directory]/refactoring/).

Sharing Custom Refactoring Operations

The purpose of Oxygen XML Editor scanning multiple locations for the XML Refactoring operations is to provide more flexibility for developers who want to share the refactoring operations with the other team members. Depending on your particular use case, you can attach the custom refactoring operations to other resources, such as frameworks or projects.

After storing custom operations, you can share them with other users by sharing the resources.

Localizing XML Refactoring Operations

Oxygen XML Editor includes localization support for the XML refactoring operations.

The translation keys for the built-in refactoring operations are located in [oxygen Installation Directory]/refactoring/i18n/translation.xml.

The localization support is also available for custom refactoring operations. The following information can be translated:

- The operation name, description, and category.
- The description of the parameters element.
- The label, description, and possibleValues for each parameter.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Editor scans the following locations to find the translation.xml files that are used to load the translation keys:

- A refactoring/i18n folder, created inside a directory that is associated to a customized *framework*.
- A i18n folder, created inside a directory that is associated to a customized *framework*.
- An i18n folder inside any specified folder. In this case, you need to [open the Preferences dialog box](#), go to **XML > XML Refactoring**, and specify the folder in the **Load additional refactoring operations from** text box.
- An i18n folder located in directories specified through the [XML Refactoring Operations Plugin Extension](#).
- The refactoring/i18n folder from the Oxygen XML Editor installation directory ([oxygen Installation Directory]/refactoring/i18n).

Example of a Refactoring Operation Descriptor File with i18n Support

```
<?xml version="1.0" encoding="UTF-8"?>

<refactoringOperationDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://oxygentools.com/app/xml_refactory
    http://oxygentools.com/app/xml_refactory/operation_descriptor.xsd"
    xmlns="http://oxygentools.com/app/xml_refactory" id="remove_text_content"
    name="${i18n(Remove_text_content)}">
    <description>${i18n(Remove_text_content_description)}</description>
    <script type="XQUERY_UPDATE" href="remove_text_content.xq"/>
    <parameters>
        <description>${i18n(parameters_description)}</description>
        <parameter label="${i18n(Element_name)}" name="element_localName" type="NC_NAME">
            <description>${i18n(Element_name_descriptor)}</description>
            <possibleValues>
                <value default="true" name="value1">${i18n(value_1)}</value>
                <value name="value2">${i18n(value_2)}</value>
            </possibleValues>
        </parameter>
    </parameters>
</refactoringOperationDescriptor>
```

Editing XSLT Stylesheets

This section explains the features of the XSLT editor.

To watch our video demonstration about basic XSLT editing and transformation scenarios in Oxygen XML Editor, go to http://oxygentools.com/demo/XSL_Editing.html.

Validating XSLT Stylesheets

Oxygen XML Editor performs the validation of XSLT documents with the help of an XSLT processor *that you can configure in the preferences pages* according to the XSLT version. For XSLT 1.0, the options are: Xalan, Saxon 6.5.5, Saxon 9.6.0.5 and *a JAXP transformer specified by the main Java class*. For XSLT 2.0, the options are: Saxon 9.6.0.5 and *a JAXP transformer specified by the main Java class*. For XSLT 3.0, the options are Saxon 9.6.0.5 and *a JAXP transformer specified by the main Java class*.

To access the *XSLT preferences* quickly, use the **Validation options** action from the **Document > Validate** menu.

Custom Validation of XSLT Stylesheets

If you need to validate an XSLT stylesheet with a validation engine that is different from the built-in engine, you can configure external engines as custom XSLT validation engines in the Oxygen XML Editor preferences. After a custom validation engine is *properly configured*, it can be applied on the current document by selecting it from the list of custom validation engines in the **Validation** toolbar drop-down list. The document is validated against the schema declared in the document.

There are two validators that are configured by default:

- **MSXML 4.0** - included in Oxygen XML Editor (Windows edition). It is associated to the XSL Editor type in *Preferences page*.
- **MSXML.NET** - included in Oxygen XML Editor (Windows edition). It is associated to the XSL Editor type in *Preferences page*.

Associate a Validation Scenario

You can validate XSLT stylesheets by using a validation scenario. To create a validation scenario, select the **Configure Validation Scenario(s)...** action from the **Document > Validate** menu, from the **Validation** toolbar drop-down list, or from the **Validate** submenu when invoking the contextual menu in the **Project** view .

You can validate an XSLT document using the engine defined in the transformation scenario, or a custom validation scenario. If you choose to validate using the engine from transformation scenario, and a transformation scenario is not associated with the current document or the engine has no validation support, the default engine is used. To set the default engine, *open the Preferences dialog box* and go to **XML > XSLT/FO/XQuery > XSLT**. The list of reusable scenarios for the appropriate document type is displayed. For more details go to *Validation Scenario*.

Editing XSLT Stylesheets in the Master Files Context

Smaller interrelated modules that define a complex stylesheet cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a function defined in a main stylesheet is not visible when you edit an included or imported module. Oxygen XML Editor provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger stylesheet structure.

You can set a main XSLT stylesheet either using the *master files support from the Project view*, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main stylesheet. In this case, it considers the current module as the main stylesheet.

The advantages of editing in the context of main file include:

- Correct validation of a module in the context of a larger stylesheet structure.
- **Content Completion Assistant** displays all components valid in the current context.
- The **Outline** displays the components collected from the entire stylesheet structure.

To watch our video demonstration about editing XSLT stylesheets in the master files context, go to <http://oxygentools.com/demo/MasterFilesSupport.html>.

Syntax Highlight

The XSL editor renders the CSS and JS scripts, and XPath expressions with dedicated coloring schemes. To customize the coloring schemes, [open the Preferences dialog box](#) and go to **Editor > Colors**.

Content Completion in XSLT Stylesheets

The items in the list of proposals offered by the **Content Completion Assistant** are context-sensitive. The proposed items are valid at the current caret position. You can enhance the list of proposals by specifying an additional schema. This schema is [defined by the user in the Content Completion / XSL preferences](#) page and can be: XML Schema, DTD, RELAX NG schema, or NVDL schema.

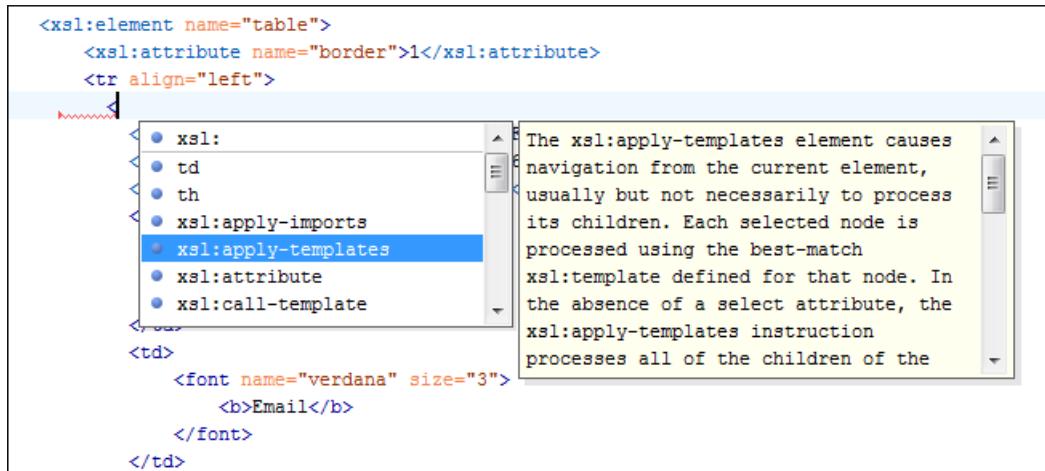


Figure 109: XSLT Content Completion Window

The **Content Completion Assistant** proposes numerous item types (such as templates, variables, parameters, keys, etc.) that are defined in the current stylesheet, and in the imported and included XSLT stylesheets. The **Content Completion Assistant** also includes [code templates that can be used to quickly insert code fragments](#) into stylesheets.

 **Note:** For XSL and XSD resources, the **Content Completion Assistant** collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to [Defining Master Files at Project Level](#).

The extension functions built in the Saxon 6.5.5 and 9.6.0.5 transformation engines are presented in the content completion list only if the Saxon namespace (<http://saxon.sf.net> for XSLT version 2.0 / 3.0 or <http://icl.com/saxon> for XSLT version 1.0) is declared and one of the following conditions is true:

- the edited file has a transformation scenario that uses as transformation engine Saxon 6.5.5 (for XSLT version 1.0), Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE (for XSLT version 2.0 / 3.0).
- the edited file has a validation scenario that uses as validation engine Saxon 6.5.5 (for version 1.0), Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE (for version 2.0 / 3.0).
- the validation engine specified in [Options](#) page is Saxon 6.5.5 (for version 1.0), Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE (for version 2.0 / 3.0).

Additionally, the Saxon-CE-specific extension functions and instructions are presented in the Content Completion Assistant's proposals list only if the `http://saxonica.com/ns/interactiveXSLT` namespace is declared.

Namespace prefixes in the scope of the current context are presented at the top of the content completion window to speed up the insertion into the document of prefixed elements.

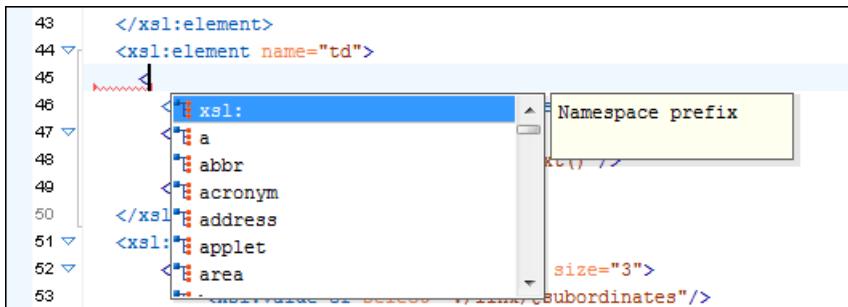


Figure 110: Namespace Prefixes in the Content Completion Window

For the common namespaces like XSL namespace (<http://www.w3.org/1999/XSL/Transform>), XML Schema namespace (<http://www.w3.org/2001/XMLSchema>) or Saxon namespace (<http://icl.com/saxon> for version 1.0, <http://saxon.sf.net/> for version 2.0 / 3.0), Oxygen XML Editor provides an easy mode to declare them by proposing a prefix for these namespaces.

Content Completion in XPath Expressions

In XSLT stylesheets, the **Content Completion Assistant** provides *all the features available in the XML editor* and also adds some enhancements. In XPath expressions used in attributes of XSLT stylesheets elements like `match`, `select` and `test`, the **Content Completion Assistant** offers the names of XPath and XSLT functions, the XSLT axes, and user-defined functions (the name of the function and its parameters). If a transformation scenario was defined and associated to the edited stylesheet, the **Content Completion Assistant** computes and presents elements and attributes based on:

- The input XML document selected in the scenario.
- The current context in the stylesheet.

The associated document is displayed in *the XSLT/XQuery Input view*.

Content completion for XPath expressions is started:

- On XPath operators detected in one of the `match`, `select` and `test` attributes of XSLT elements: ', ', /, //, (, [, |, :, ::, \$
- For attribute value templates of non-XSLT elements, that is the { character when detected as the first character of the attribute value.
- On request, if the combination **Ctrl Space (Command Space on OS X)** is pressed inside an edited XPath expression.

The items presented in the content completion window are dependent on:

- The context of the current XSLT element.
- The XML document associated with the edited stylesheet in the stylesheet transformation scenario.
- The XSLT version of the stylesheet (1.0, 2.0, or 3.0).



Note: The XSLT 3.0 content completion list of proposals includes specific elements and attributes for the 3.0 version.

For example, if the document associated with the edited stylesheet is:

```

<personnel>
  <person id="Big.Boss">
    <name>
      <family>Boss</family>
      <given>Big</given>
    </name>
    <email>chief@oxygenvml.com</email>
    <link subordinates="one.worker"/>
  </person>
  <person id="one.worker">
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>

```

```

<email>one@oxygencxml.com</email>
<link manager="Big.Boss"/>
</person>
</personnel>

```

If you enter an `xsl:template` element using the content completion assistant, the following actions are triggered:

- The `match` attribute is inserted automatically.
- The cursor is placed between the quotes.
- The XPath **Content Completion Assistant** automatically displays a popup window with all the XSLT axes, XPath functions and elements and attributes from the XML input document that can be inserted in the current context.

The set of XPath functions depends on the XSLT version declared in the root element `xsl:stylesheet`: 1.0, 2.0 or 3.0.

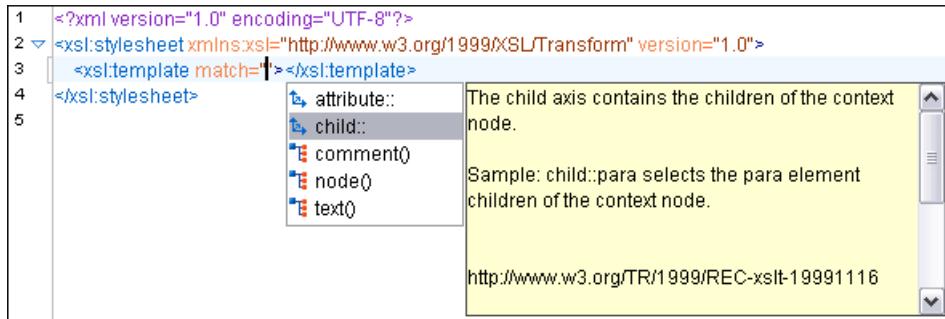


Figure 111: Content Completion in the `match` Attribute

If the cursor is inside the `select` attribute of an `xsl:for-each`, `xsl:apply-templates`, `xsl:value-of` or `xsl:copy-of` element the content completion proposals depend on the path obtained by concatenating the XPath expressions of the parent XSLT elements `xsl:template` and `xsl:for-each` as shown in the following figure:

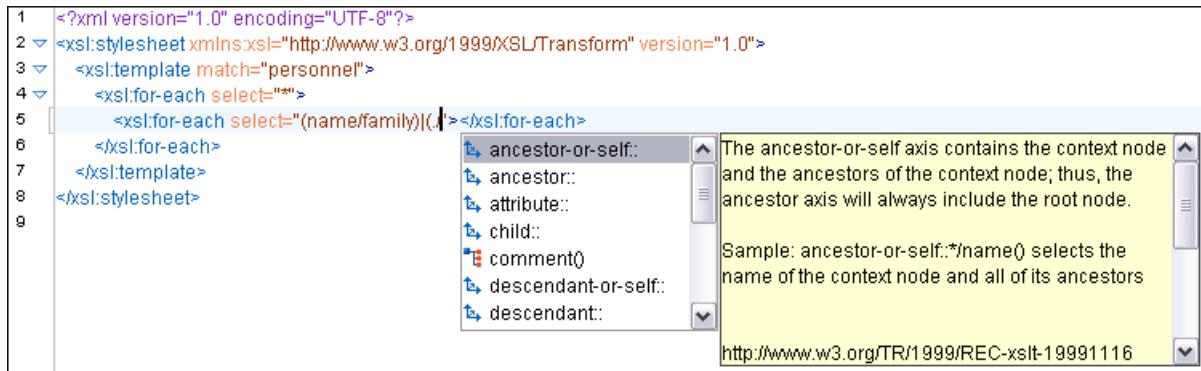


Figure 112: Content Completion in the `select` Attribute

Also XPath expressions typed in the `test` attribute of an `xsl:if` or `xsl:when` element benefit of the assistance of the content completion.

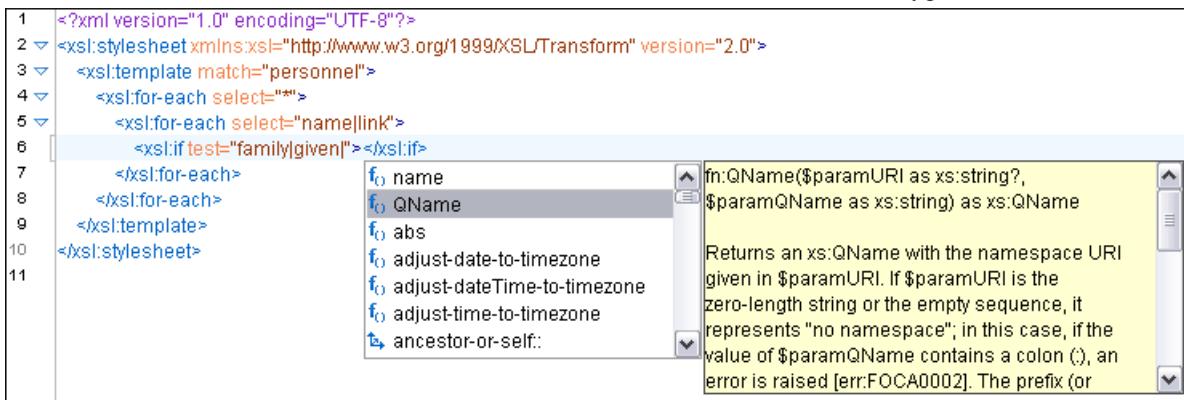


Figure 113: Content Completion in the test Attribute

XSLT variable references are easier to insert in XPath expressions with the help of the content completion popup triggered by the \$ character which signals the start of such a reference in an XPath expression.

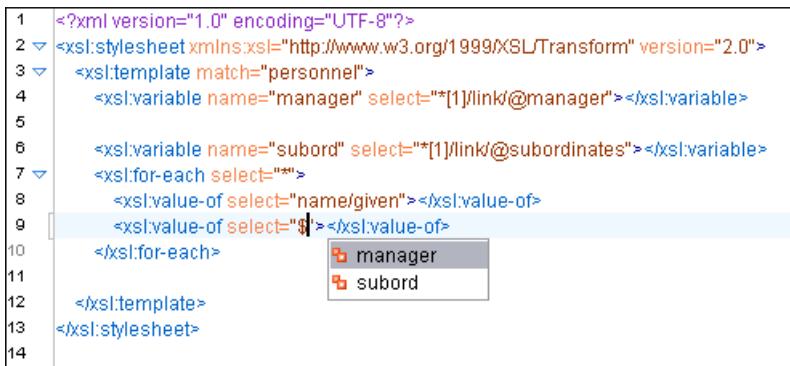


Figure 114: Content Completion in the test Attribute

If the { character is the first one in the value of the attribute, the same **Content Completion Assistant** is available also in attribute value templates of non-XSLT elements.

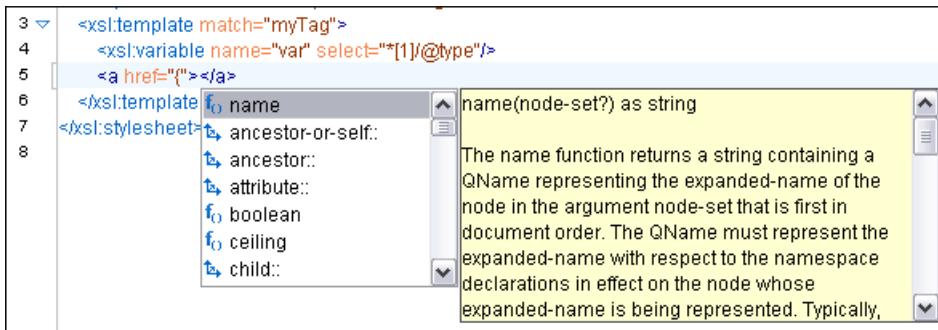


Figure 115: Content Completion in Attribute Value Templates

The time delay *configured in Preferences* page for all content completion windows is applied also for the XPath expressions content completion window.

Tooltip Helper for the XPath Functions Arguments

When editing the arguments of an XPath/XSLT function, Oxygen XML Editor tracks the current entered argument by displaying a tooltip containing the function signature. The currently edited argument is highlighted with a bolder font.

When moving the caret through the expression, the tooltip is updated to reflect the argument found at the caret position.

We want to concatenate the absolute values of two variables, named *v1* and *v2*.

```
<xsl:template match="/">
  <xsl:value-of select="concat(abs($v1), abs($v2))"></xsl:value-of>
</xsl:template>
```

When moving the caret before the first `abs` function, Oxygen XML Editor identifies it as the first argument of the `concat` function. The tooltip shows in bold font the following information about the first argument:

- Its name is `$arg1`.
- Its type is `xdt:anyAtomicType`.
- It is optional (note the `?` sign after the argument type).

The function takes also other arguments, having the same type, and returns a `xs:string`.

```
name="concat($arg1 as xdt:anyAtomicType?, $arg2 as xdt:anyAtomicType?, ...)" as xs:string
@match="r"
@select="concat(abs($v1), abs($v2))"></xsl:value-of>
</xsl:template>
</stylesheet>
```

Figure 116: XPath Tooltip Helper - Identify the concat Function's First Argument

Moving the caret on the first variable `$v1`, the editor identifies the `abs` as context function and shows its signature:

```
name="v2" select="abs($arg as numeric?)" as numeric?
@match="r"
@select="concat(abs($v1), abs($v2))"></xsl:value-of>
</xsl:template>
</stylesheet>
```

Figure 117: XPath Tooltip Helper - Identify the abs Function's Argument

Further, clicking the second `abs` function name, the editor detects that it represents the second argument of the `concat` function. The tooltip is repainted to display the second argument in bold font.

```
name="concat($arg1 as xdt:anyAtomicType?, $arg2 as xdt:anyAtomicType?, ...)" as xs:string
@match="r"
@select="concat(abs($v1), abs($v2))"></xsl:value-of>
</xsl:template>
</stylesheet>
```

Figure 118: XPath Tooltip Helper - Identify the concat Function's Second Argument

The tooltip helper is available also in the XPath toolbar and the **XPath Builder** view.

The XSLT/XQuery Input View

The structure of the XML document associated to the edited XSLT stylesheet, or the structure of the source documents of the edited XQuery is displayed in a tree form in a view called **XSLT/XQuery Input**. The tree nodes represent the elements of the documents.

The XSLT Input View

If you click a node, the corresponding template from the stylesheet is highlighted. A node can be dragged from this view and dropped in the editor area for quickly inserting `xsl:template`, `xsl:for-each`, or other XSLT elements that have the `match/select/test` attribute already completed. The value of the attribute is the correct XPath expression that refers to the dragged tree node. This value is based on the current editing context of the drop spot.

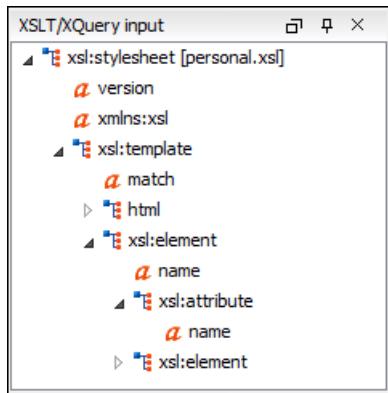


Figure 119: XSLT Input View

For example, for the following XML document:

```
<personnel>
  <person id="Big.Boss">
    <name>
      <family>Boss</family>
      <given>Big</given>
    </name>
    <email>chief@oxygenvxml.com</email>
    <link subordinates="one.worker"/>
  </person>
  <person id="one.worker">
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>
    <email>one@oxygenvxml.com</email>
    <link manager="Big.Boss"/>
  </person>
</personnel>
```

and the following XSLT stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
  <xsl:template match="personnel">
    <xsl:for-each select="*"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

if you drag the given element and drop it inside the xsl:for-each element, the following popup menu is displayed:

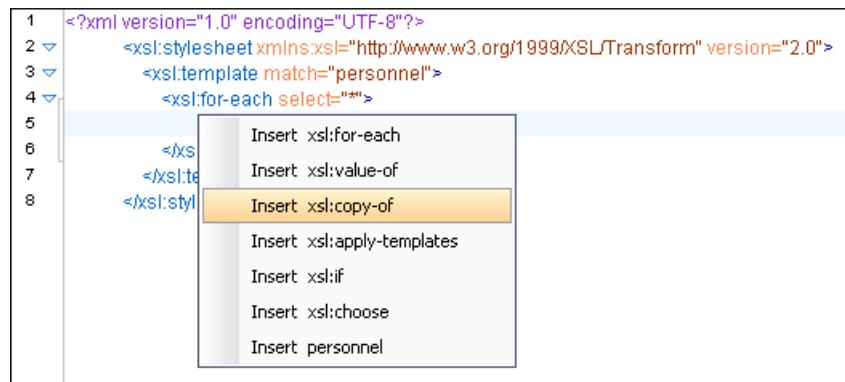


Figure 120: XSLT Input Drag and Drop Popup Menu

Select for example **Insert xsl:value-of** and the result document is:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3    <xsl:template match="personnel">
4      <xsl:for-each select="*">
5        <xsl:value-of select="name/given"/>
6      </xsl:for-each>
7    </xsl:template>
8  </xsl:stylesheet>

```

Figure 121: XSLT Input Drag and Drop Result

The XSLT Outline View

The **XSLT Outline** view displays the list of all the components (templates, attribute-sets, character-maps, variables, functions, keys, outputs) from both the edited stylesheet and its imports or includes. For XSL and XSD resources, the **Outline** view collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to [Defining Master Files at Project Level](#). To enable the **Outline** view, go to **Window > Show View > Outline**.

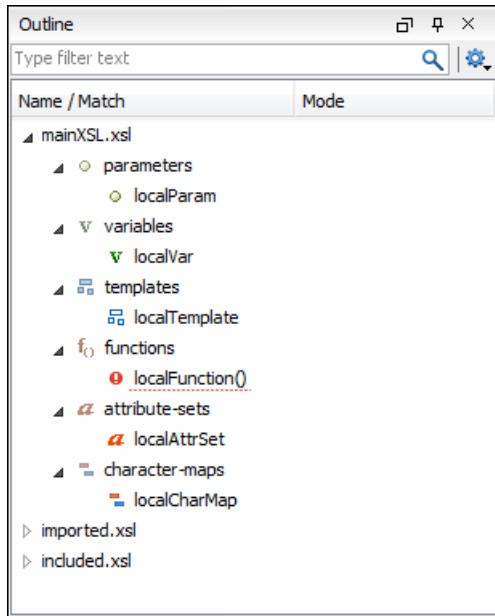


Figure 122: The XSLT Outline View

The following actions are available in the **Settings** menu on the Outline view toolbar:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches;

Selection update on caret move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret moves or the changes in the XSLT editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Show XML structure

Displays the XML document structure in a tree-like structure.

Show all components

Displays all components that were collected starting from the main file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/type

The stylesheet components can be grouped by location and type.

Show components

Shows the define patterns collected from the current document.

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).

Configure displayed attributes

Displays the [XML Structured Outline preferences page](#).

The following contextual menu actions are available:

Append Child

Displays a list of elements that can be inserted as children of the current element.

Insert Before

Displays a list of elements that can be inserted as siblings of the current element, before the current element.

Insert After

Displays a list of elements that can be inserted as siblings of the current element, after the current element.

Toggle Comment

Comments/uncomments the currently selected element.

Remove (Delete)

Removes the selected item from the stylesheet.

Search References **Ctrl Shift R** (**Command Shift R** on OS X)

Searches all references of the item found at current cursor position in the defined scope, if any. See [Finding XSLT References and Declarations](#) for more details.

Search References in...

Searches all references of the item found at current cursor position in the specified scope. See [Finding XSLT References and Declarations](#) for more details.

Component Dependencies

Allows you to see the dependencies for the current selected component. See [Component Dependencies View](#) for more details.

Rename Component in...

Renames the selected component. See [XSLT Refactoring Actions](#) for more details.

The stylesheet components information is presented on two columns: the first column presents the name and `match` attributes, the second column the `mode` attribute. If you know the component name, `match` or `mode`, you can search it in the **Outline** view by typing one of these pieces of information in the filter text field from the top of the view or directly on the tree structure. When you type de component name, `match` or `mode` in the text field, you can switch to the tree structure using:

- keyboard arrow keys
- **Enter** key
- **Tab** key
- **Shift-Tab** key combination

To switch from tree structure to the filter text field, you can use **Tab** and **Shift-Tab**.

 **Tip:** The search filter is case insensitive. The following wildcards are accepted:

- * - any string
- ? - any character
- , - patterns separator

If no wildcards are specified, the string to search is used as a partial match (like `*textToFind*`).

On the XSLT **Outline** view, you have some contextual actions like: **Edit Attributes**, **Cut**, **Copy**, **Delete**.

The **Outline** content and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

Oxygen XML Editor allows you to sort the components of the tree in the **Outline** view.

 **Note:** Sorting groups in the **Outline** view is not supported.

Oxygen XML Editor has a predefined order of the groups in the **Outline** view:

- for location, the names of the files are sorted alphabetically. The main file is the one you are editing and it is located at the top of the list
- for type, the order is: parameters, variables, templates, functions, set attributes, character-map

 **Note:** When no grouping is available and the table is not sorted, Oxygen XML Editor sorts the components depending on their order in the document. Oxygen XML Editor also takes into account the name of the file that the components are part of.

XSLT Stylesheet Documentation Support

Oxygen XML Editor offers built-in support for documenting XSLT stylesheets. If the expanded *QName* of the element has a non-null namespace URI, the `xsl:stylesheet` element may contain any element not from the XSLT namespace. Such elements are referenced as user-defined data elements. Such elements can contain the documentation for the stylesheet and its elements (top-level elements whose names are in the XSLT namespace). Oxygen XML Editor offers its own XML schema that defines such documentation elements. The schema is named `stylesheet_documentation.xsd` and can be found in `[OXYGEN_DIR]/frameworks/stylesheet_documentation`. The user can also specify a custom schema in [XSL Content Completion options](#).

When content completion is invoked inside an XSLT editor by pressing **Ctrl Space (Command Space on OS X)**, it offers elements from the XSLT documentation schema (either the built-in one or one specified by user).

In **Text** mode, to add documentation blocks while editing use the **Add component documentation** action available in the contextual menu.

In **Author** mode, the following stylesheet documentation actions are available in the contextual menu, **Component Documentation** submenu:

- **Add component documentation** - Adds documentation blocks for the component at caret position.

- **Paragraph** - Inserts a new documentation paragraph.
- **Bold** - Makes the selected documentation text bold.
- **Italic** - Makes the selected documentation text italic.
- **List** - Inserts a new list.
- **List Item** - Inserts a list item.
- **Reference** - Inserts a documentation reference.

If the caret is positioned inside the `xsl:stylesheet` element context, documentation blocks are generated for all XSLT elements. If the caret is positioned inside a specific XSLT element (like a template or a function), a documentation block is generated for that element only.

Example of a documentation block using Oxygen XML Editor built-in schema

```
<xd:doc>
  <xd:desc>
    <xd:p>Search inside parameter <xd:i>string</xd:i> for the last occurrence of parameter
    <xd:i>searched</xd:i>. The substring starting from the 0 position to the identified last
    occurrence will be returned. <xd:ref name="f:substring-after-last" type="function"
    xmlns:f="http://www.oxygenxml.com/doc/xsl/functions">See also</xd:ref></xd:p>
  </xd:desc>
  <xd:param name="string">
    <xd:p>String to be analyzed</xd:p>
  </xd:param>
  <xd:param name="searched">
    <xd:p>Marker string. Its last occurrence will be identified</xd:p>
  </xd:param>
  <xd:return>
    <xd:p>A substring starting from the beginning of <xd:i>string</xd:i> to the last
    occurrence of <xd:i>searched</xd:i>. If no occurrence is found an empty string will be
    returned.</xd:p>
  </xd:return>
</xd:doc>
```

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Editor to generate detailed documentation in HTML format for the elements (top-level elements whose names are in the XSLT namespace) of an XSLT stylesheet. You are able to select what XSLT elements to include in the generated documentation and also the level of details to present for each of them. The elements are hyperlinked. To generate documentation in a custom format, other than HTML, you can edit the XSLT stylesheet used to generate the documentation, or create your own stylesheet.

To open the **XSLT Stylesheet Documentation** dialog box, select **XSLT Stylesheet Documentation...** from the **Tools > Generate Documentation** menu or from the **Generate Documentation** submenu in the contextual menu of the **Project** view.

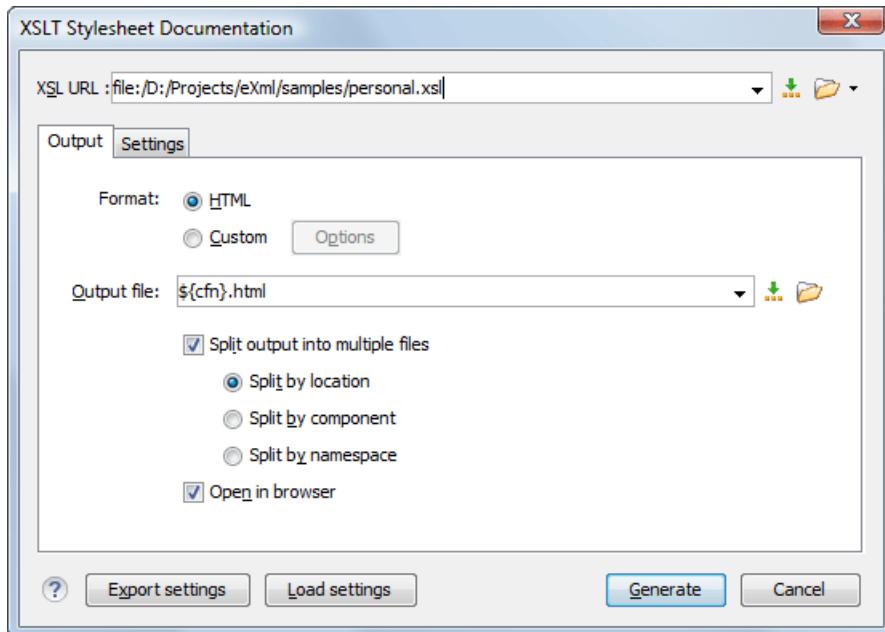


Figure 123: The Output Panel of the XSLT Stylesheet Documentation Dialog Box

The **XSL URL** field of the dialog box must contain the full path to the XSL Stylesheet file you want to generate documentation for. The stylesheet can be either a local or a remote one. You can also specify the path of the stylesheet using editor variables.

You can choose to split the output into multiple files using different split criteria. For large XSLT stylesheets being documented, choosing a different split criterion may generate smaller output files providing a faster documentation browsing.

The available split criteria are:

- by location - Each output file contains the XSLT elements from the same stylesheet.
- by namespace - Each output file contains information about elements with the same namespace.
- by component - Each output file contains information about one stylesheet XSLT element.

You can export the settings of the **XSLT Stylesheet Documentation** dialog box to an XML file by pressing the **Export settings** button. With the exported settings file, you can generate the same *documentation from the command-line interface*.

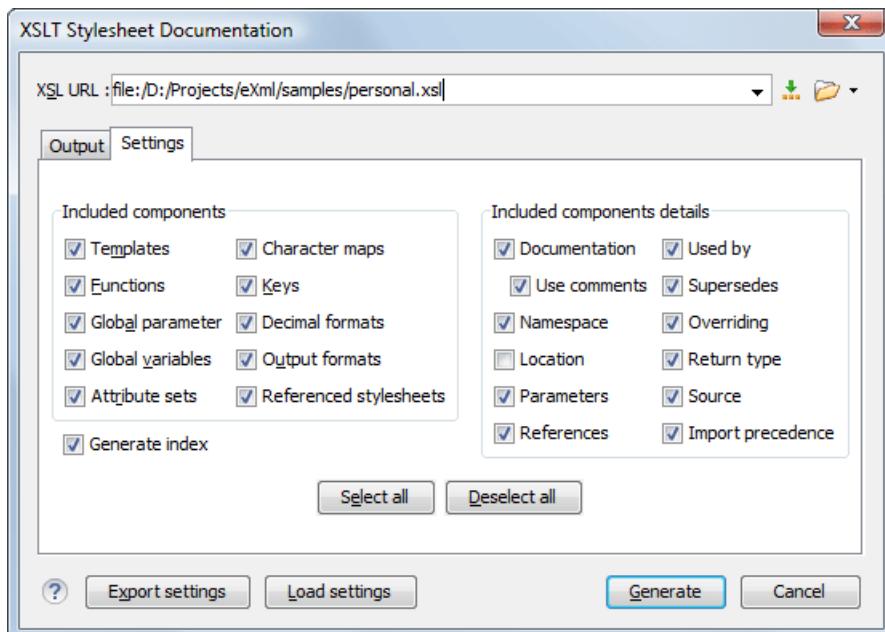


Figure 124: The Settings Panel of the XSLT Stylesheet Documentation Dialog Box

When you generate documentation for an XSLT stylesheet you can choose what XSLT elements to include in the output (templates, functions, global parameters, global variables, attribute sets, character maps, keys, decimal formats, output formats, XSLT elements from referenced stylesheets) and the details to include in the documentation:

- **Documentation** - Shows the documentation for each XSLT element. For HTML format, the user-defined data elements that are recognized and transformed in documentation blocks of the XSLT elements they precede, are the ones from the following schemas:
 - Oxygen XML Editor built-in XSLT documentation schema.
 - A subset of DocBook 5 elements. The recognized elements are: `section`, `sect1` to `sect5`, `emphasis`, `title`, `ulink`, `programlisting`, `para`, `orderedlist`, `itemizedlist`.
 - A subset of DITA elements. The recognized elements are: `concept`, `topic`, `task`, `codeblock`, `p`, `b`, `i`, `ul`, `ol`, `pre`, `s1`, `sli`, `step`, `steps`, `li`, `title`, `xref`.
 - Full XHTML 1.0 support.
 - XSLStyle documentation environment. XSLStyle uses DocBook or DITA languages inside its own user-defined data elements. The supported DocBook and DITA elements are the ones mentioned above.
 - Doxsl documentation framework. Supported elements are : `codefrag`, `description`, `para`, `docContent`, `documentation`, `parameter`, `function`, `docSchema`, `link`, `list`, `listitem`, `module`, `parameter`, `template`, `attribute-set`;

Other XSLT documentation blocks that are not recognized will just be serialized inside an HTML `pre` element. You can change this behavior by using a [custom format](#) instead of the built-in [HTML format](#) and providing your own XSLT stylesheets.

- **Use comments** - Controls whether the comments that precede an XSLT element is treated as documentation for the element they precede. Comments that precede or succeed the `xsl:stylesheet` element, are treated as documentation for the whole stylesheet. Please note that comments that precede an import or include directive are not collected as documentation for the imported/included module. Also comments from within the body of the XSLT elements are not collected at all.
- **Namespace** - Shows the namespace for named XSLT elements.
- **Location** - Shows the stylesheet location for each XSLT element.
- **Parameters** - Shows parameters of templates and functions.
- **References** - Shows the named XSLT elements that are referenced from within an element.
- **Used by** - Shows the list of all the XSLT elements that reference the current named element.

- **Supersedes** - Shows the list of all the XSLT elements that are superseded the current element.
- **Overriding** - Shows the list of all the XSLT elements that override the current element.
- **Return type** - Shows the return type of the function.
- **Source** - Shows the text stylesheet source for each XSLT element.
- **Import precedence** - Shows the computed import precedence as declared in XSL transformation specifications.
- **Generate index** - Creates an index with all the XSLT elements included in the documentation.
- **Load settings / Export settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

Generate Documentation in HTML Format

The generated documentation looks like:

The screenshot shows the Oxygen XML Editor's Stylesheet documentation feature. On the left, there's a 'Table of Contents' pane where you can group contents by location, namespace, or component type. The main pane displays the documentation for the 'Main stylesheet xslDocHtml.xsl'. It includes sections for 'Documentation' (with a 'Description' sub-section), 'Imported modules', and 'Templates'. A right-hand sidebar titled 'Showing:' lists several checkboxes: Documentation (checked), Parameters (checked), Used by (checked), References (checked), Imported modules (checked), and Source (checked). Below this is a 'Close' button. In the bottom section, there's a detailed view for a template named 'createJsIdsArray'. This view includes sections for 'Documentation' (with a 'Description' sub-section), 'Parameters' (with 'arrayName' listed), 'Nodes' (with a description of generating IDs for XSLT elements), 'Namespace' (listing 'No namespace'), 'Used by' (listing 'Variables' and 'Function'), and 'References' (listing 'getDId(\$node as item())').

Figure 125: XSLT Stylesheet Documentation Example

The generated documentation includes the following:

- Table of Contents - You can group the contents by namespace, location, or component type. The XSLT elements from each group are sorted alphabetically (named templates are presented first and the match ones second).
- Information about main, imported, and included stylesheets - This information consists of:
 - XSLT modules included or imported by the current stylesheet
 - the XSLT stylesheets where the current stylesheet is imported or included
 - the stylesheet location

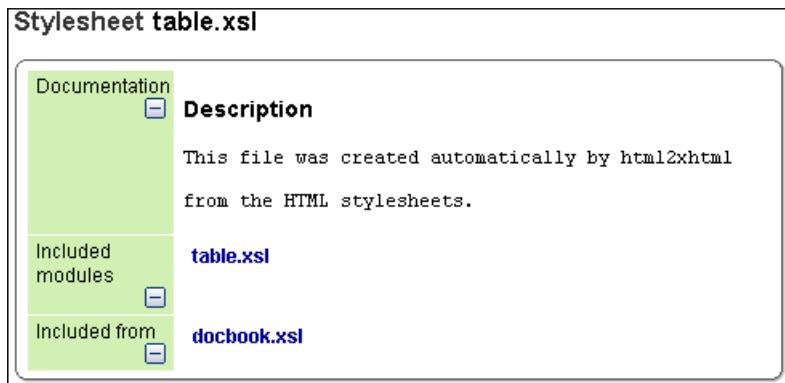


Figure 126: Information About an XSLT Stylesheet

If you choose to split the output into multiple files, the table of contents is displayed in the left frame. The contents are grouped using the same criteria as the split.

After the documentation is generated, you can collapse details for some stylesheet XSLT elements using the **Showing** view.

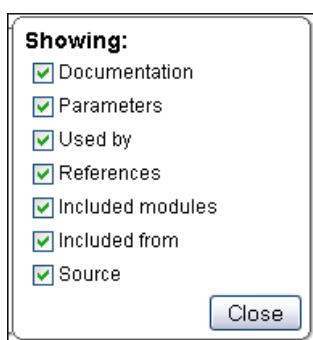


Figure 127: The Showing View

For each element included in the documentation, the section presents the element type followed by the element name (value of the name or match attribute for match templates).

Function func:substring-before-last

Documentation							
Description Get the substring before the last occurrence of the given substring							
Parameters							
string The string in which to search							
searched The string to search							
Return							
The substring starting from the start of the string to the index of the last occurrence of searched							
Namespace	http://www.oxygenxml.com/doc/xsl/functions						
Type	xs:string						
Used by	Template Nindex Function Variable indexFile						
References	Function substring-before-last(\$string as item(), \$searched as item())						
Parameters	<table border="1"> <tr> <td>QName</td> <td>Namespace</td> </tr> <tr> <td>searched</td> <td>No namespace</td> </tr> <tr> <td>string</td> <td>No namespace</td> </tr> </table>	QName	Namespace	searched	No namespace	string	No namespace
QName	Namespace						
searched	No namespace						
string	No namespace						
Import precedence	7						
Source	<pre><xsl:function as="xs:string" name="func:substring-before-last"> <xsl:param name="string"/> <xsl:param name="searched"/> <xsl:variable name="toReturn"> <xsl:choose> <xsl:when test="contains(\$string, \$searched)"> <xsl:variable name="before" select="substring-before(\$string, \$searched)"/> <xsl:variable name="rec" select="func:substring-before-last(substring-after(\$string, \$searched), \$searched)"/> <xsl:otherwise> \$string </xsl:otherwise> </xsl:choose> </xsl:variable> </xsl:function></pre>						

Figure 128: Documentation for an XSLT Element

Generate Documentation in a Custom Format

XSLT stylesheet documentation can be also generated in a custom format. You can choose the format from the [XSLT Stylesheet Documentation dialog box](#). Specify your own stylesheet to transform the intermediary XML generated in the documentation process. You must write your stylesheet based on the schema xs1DocSchema.xsd from [OXYGEN_DIR]/frameworks/stylesheet_documentation. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in [OXYGEN_DIR]/frameworks/stylesheet_documentation/xsl.

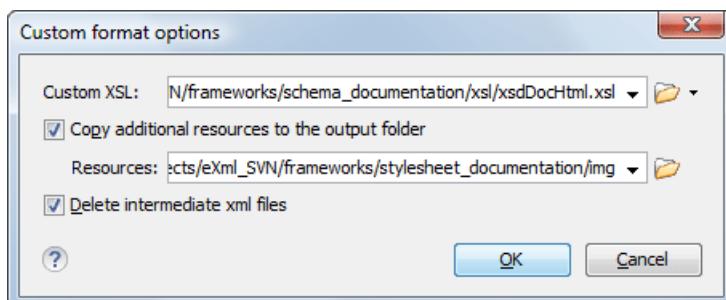


Figure 129: The Custom Format Options Dialog Box

When using a custom format, you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating Documentation From the Command Line Interface

You can export the settings of the **XSLT Stylesheet Documentation** dialog box to an XML file by pressing the **Export settings** button. With the exported settings file, you can generate the same documentation from the command line by running the script `stylesheetDocumentation.bat` (on Windows) / `stylesheetDocumentation.sh` (on OS X / Unix / Linux) located in the Oxygen XML Editor installation folder. The script can be integrated in an external batch process launched from the command-line interface.

The command-line parameter of the script is the relative path to the exported XML settings file. The files which are specified with relative paths in the exported XML settings are resolved relative to the script directory.

Example of an XML Configuration File

```
<serialized>
  <map>
    <entry>
      <String xml:space="preserve">xsd.documentation.options</String>
      <xsdDocumentationOptions>
        <field name="outputFile">
          <String xml:space="preserve">${cfn}.html</String>
        </field>
        <field name="splitMethod">
          <Integer xml:space="preserve">1</Integer>
        </field>
        <field name="openOutputInBrowser">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="format">
          <Integer xml:space="preserve">1</Integer>
        </field>
        <field name="customXSL">
          <null/>
        </field>
        <field name="deleteXMLFiles">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeIndex">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeGlobalElements">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeGlobalAttributes">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeLocalElements">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeLocalAttributes">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeSimpleTypes">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeComplexTypes">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeGroups">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeAttributesGroups">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeRedefines">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeReferencedSchemas">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="detailsDiagram">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="detailsNamespace">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="detailsLocation">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
```

```

<field name="detailsType">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsTypeHierarchy">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsModel">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsChildren">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsInstance">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsUsedby">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsProperties">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsFacets">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsAttributes">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsIdentityConstr">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsEscapeAnn">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsSource">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsAnnotations">
    <Boolean xml:space="preserve">true</Boolean>
</field>
</xsdDocumentationOptions>
</entry>
</map>
</serialized>
```

Finding XSLT References and Declarations

The following search actions related with XSLT references and declarations are available from the **Search** submenu of the contextual menu and from the **Document > References** menu:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined but the currently edited resource is not part of the range of determined resources, a warning dialog box is displayed that allows you to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when a scope is defined.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined but the current edited resource is not part of the range of resources determined by this scope, a warning dialog box is displayed that allows you to define another search scope.
- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when a scope is defined.
- **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

The following action is available from the contextual menu and the **Document > Schema** menu:

- **Show Definition** - Moves the cursor to the location of the definition of the current item.
- Note:** You can also use the **Ctrl Click (Command Click on OS X)** shortcut on a reference to display its definition.

Highlight Component Occurrences

When a component (for example variable or named template) is found at current cursor position, Oxygen XML Editor performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document.

-  **Note:** Oxygen XML Editor also supports occurrences highlight for template modes.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is enabled by default. To configure it, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. A search can also be triggered with the **Search > Search Occurrences in File (Ctrl Shift U (Command Shift U on OS X))** contextual menu action. Matches are displayed in separate tabs of the **Results** view.

XSLT Refactoring Actions

Oxygen XML Editor offers a set of actions that allow changing the structure of an XSLT stylesheet without changing the results of running it in an XSLT transformation. Depending on the selected text, the following refactoring actions are available from **Refactoring** submenu from the contextual menu of the current editor and the **Document > Refactoring** menu:

-  **Extract template...** - Extracts the selected XSLT instructions sequence into a new template. Opens a dialog that allows you to specify the name of the new template to be created. The possible changes to perform on the document can be previewed before altering the document. After pressing OK, the template is created and the selection is replaced with a `<xsl:call-template>` instruction referencing the newly created template.



Note: This action is available only when the selection contains well-formed elements.



Note: The newly created template is indented and its name is highlighted in the `<xsl:call-template>` element.

-  **Move to another stylesheet...** - Allows you to move one or more XSLT global components (templates, functions or parameters) to another stylesheet. Active only when these components are selected. Follow these steps:
 - execute the **Move to another stylesheet** action. You will be prompted to select the destination stylesheet, which can be: a new stylesheet or an already existing one.
 - press the **Choose** button to navigate to the destination stylesheet file. Oxygen XML Editor will automatically check if the destination stylesheet is already contained by the hierarchy of the current stylesheet. If it is not contained, choose if the destination stylesheet will be referenced (imported or included) or not from the current stylesheet. The following options are available:
 - **Include** - the current stylesheet will use an `xsl:include` instruction to reference the destination stylesheet.
 - **Import** - the current stylesheet will use an `xsl:import` instruction to reference the destination stylesheet.
 - **None** - there will be created no relation between the current and destination stylesheets.
 - press the **Move** button to move the components to the destination stylesheet. After the action's execution, the moved components are highlighted in the destination stylesheet.
- **Convert attributes to xsl:attributes** - Converts the attributes from the selected element and represents each of them with an `<xsl:attribute>` instruction. For example, from the following element:

```
<person id="Big{test}Boss"/>
```

you obtain:

```
<person>
  <xsl:attribute name="id">
    <xsl:text>Big</xsl:text>
    <xsl:value-of select="test"/>
    <xsl:text>Boss</xsl:text>
```

```
</xsl:attribute>
</person>
```

- **Convert xsl:if into xsl:choose/xsl:when** - Converts an `xsl:if` block to an `xsl:when` block surrounded by an `xsl:choose` element. For example, the following block:

```
<xsl:if test="a">
  <!-- XSLT code -->
</xsl:if>
```

is converted to:

```
<xsl:choose>
  <xsl:when test="a">
    <!-- XSLT code -->
  </xsl:when>
  <xsl:otherwise>
    |
    </xsl:otherwise>
  </xsl:choose>
```

where the `|` character is the current caret position.

- **Extract local variable** - Allows you to create a new local variable by extracting the selected XPath expression. After creating the new local variable before the current element, Oxygen XML Editor allows you to edit in-place the variable's name.



Note: The action is active on a selection made inside an attribute that contains an XPath expression.

- **Extract global variable** - Allows you to create a new global variable by extracting the selected XPath expression. After creating the new global variable, Oxygen XML Editor allows you to edit in-place the variable's name.



Note: The action is active on a selection made inside an attribute that contains an XPath expression.



Note: Oxygen XML Editor checks if the selected expression depends on local variables or parameters that are not available in the global context where the new variable is created.

- **Extract template parameter** - Allows you to create a new template parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Editor allows you to edit in-place its name.



Note: The action is active on a selection made inside an attribute that contains an XPath expression.

- **Extract global parameter** - Allows you to create a new global parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Editor allows you to edit in-place its name.



Note: The action is active on a selection made inside an attribute that contains an XPath expression.



Note: Oxygen XML Editor checks if the selected expression depends on local variables or parameters that are not available in the global context where the new parameter is created.

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the `Esc` or `Enter` key on your keyboard.

- **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

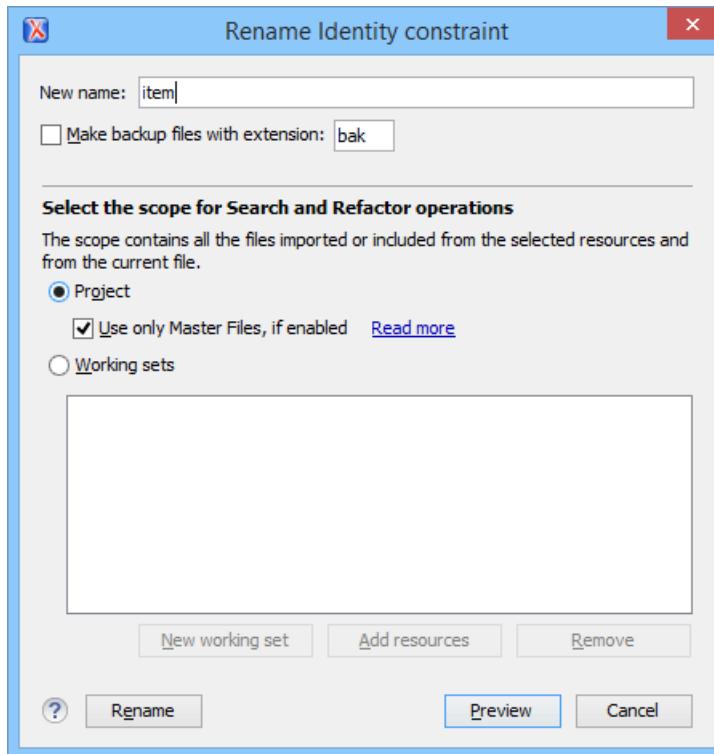


Figure 130: Rename Identity Constraint Dialog Box



Note: These refactoring actions are also proposed by the *Quick Assist support*.

To watch our video demonstration about XSLT refactoring, go to http://oxygentools.com/demo/XSL_Refactoring.html.

XSLT Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a stylesheet. To open this view, go to **Window > Show View > Resource Hierarchy/Dependencies**.

If you want to see the hierarchy of a stylesheet, select the desired stylesheet in the project view and choose **Resource Hierarchy** from the contextual menu.

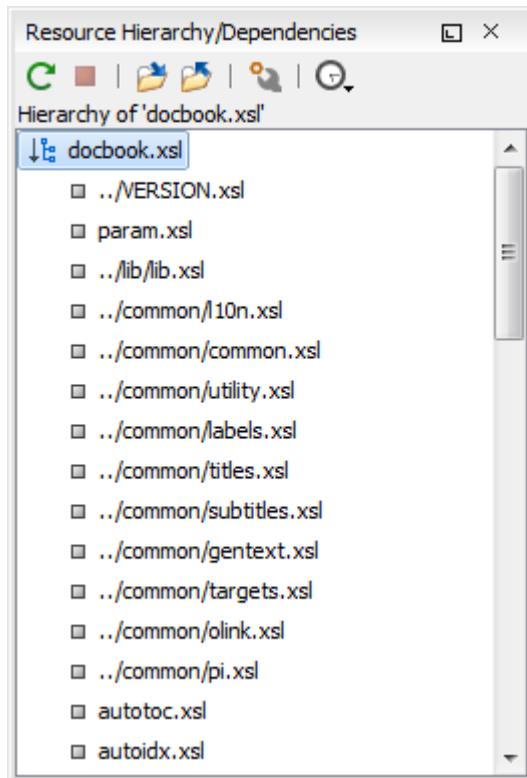


Figure 131: Resource Hierarchy/Dependencies View - Hierarchy for docbook.xsl

If you want to see the dependencies of a stylesheet, select the desired stylesheet in the project view and choose **Resource Dependencies** from the contextual menu.

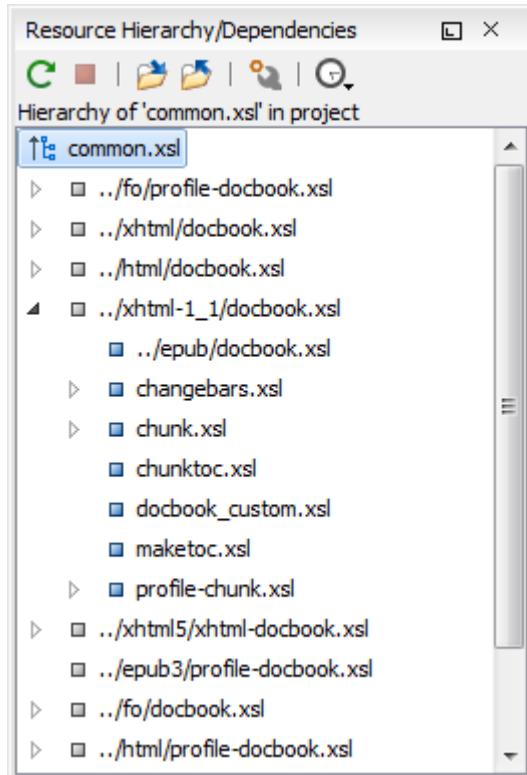


Figure 132: Resource Hierarchy/Dependencies View - Dependencies for common.xsl

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the  **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

 **Tip:** When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .

Moving/Renaming XSLT Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Component Dependencies View

The Component Dependencies view allows you to see the dependencies for a selected XSLT component. You can open the view from **Window > Show View > Component Dependencies**.

If you want to see the dependencies of an XSLT component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named components (templates, variables, parameters, attribute sets, keys, functions, outputs).

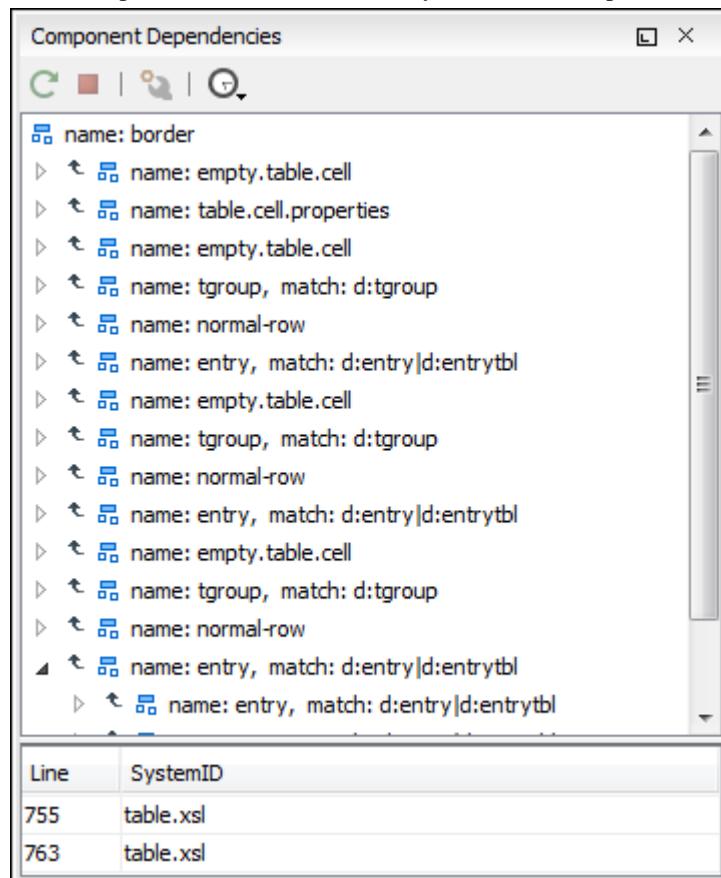


Figure 133: Component Dependencies View - Hierarchy for table.xsl

In the Component Dependencies view you have several actions in the toolbar:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

History

Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

 **Tip:** If a component contains multiple references to another, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon .

XSLT Quick Assist Support

The **Quick Assist** support helps you to rapidly access search and refactoring actions. If one or more actions are available in the current context, they are accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the **Quick Assist** menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

Two categories of actions are available in the **Quick Assist** menu:

- Actions available on a selection made inside an attribute that contains an XPath expression:
 -  **Extract template** - Extracts the selected XSLT instructions sequence into a new template.
 -  **Move to another stylesheet** - Allows you to move one or more XSLT global components (templates, functions or parameters) to another stylesheet.
 - **Extract local variable** - Allows you to create a new local variable by extracting the selected XPath expression.
 - **Extract global variable** - Allows you to create a new global variable by extracting the selected XPath expression.
 - **Extract template parameter** - Allows you to create a new template parameter by extracting the selected XPath expression.
 - **Extract global parameter** - Allows you to create a new global parameter by extracting the selected XPath expression.

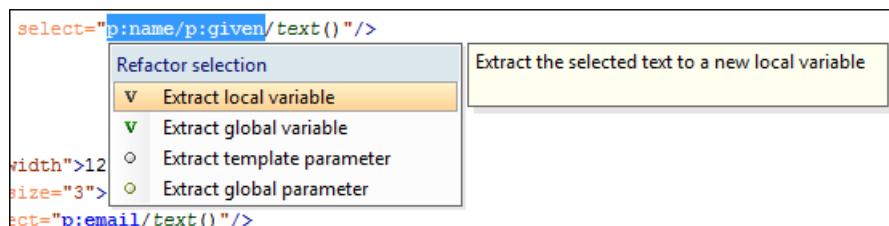


Figure 134: XSLT Quick Assist Support - Refactoring Actions

- actions available when the cursor is positioned over the name of a component:

Rename Component in...

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

 **Search References**

Searches all references of the component in a predefined scope.

 **Component Dependencies**

Searches the component dependencies in a predefined scope.

 **Change Scope...**

Configures the scope that will be used for future search or refactor operations.

 **Rename Component**

Allows you to rename the current component in-place.

 **Search Occurrences**

Searches all occurrences of the component within the current file.

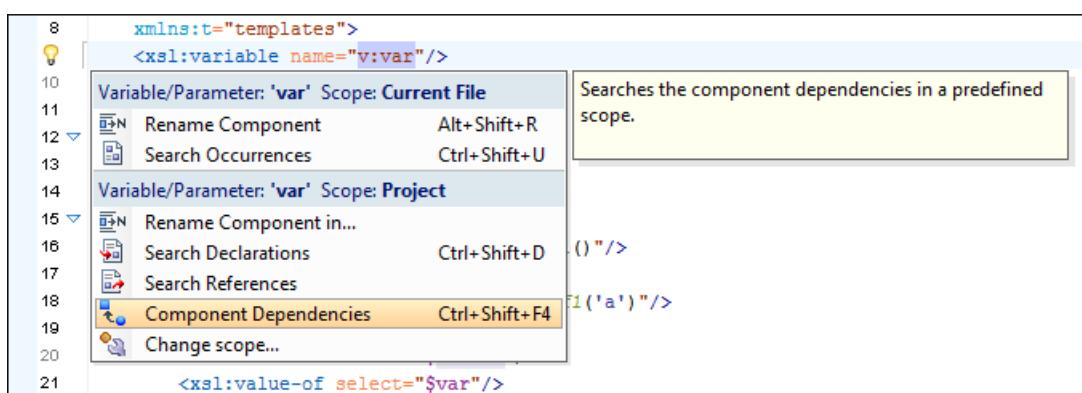


Figure 135: XSLT Quick Assist Support - Component Actions

XSLT Quick Fix Support

The Oxygen XML Editor Quick Fix support helps you resolve different errors that appear in a stylesheet by offering quick fixes to problems like a missing template, misspelled template name, missing function or references to an undeclared variable or parameter.

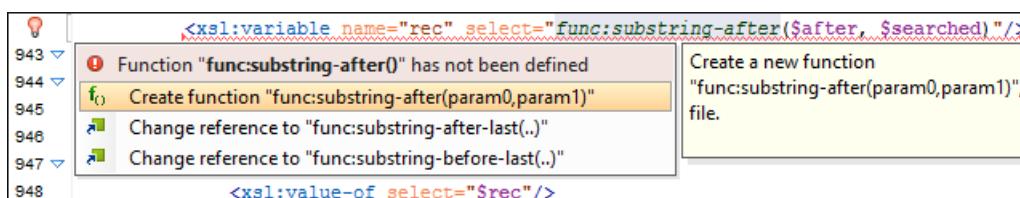


Figure 136: XSLT Functions Quick Fix

To activate the feature, when Oxygen XML Editor finds a validation error in an XSLT stylesheet, place the caret in the highlighted area of text. If Oxygen XML Editor can provide a quick fix for that error, the  icon is displayed in the left side stripe. When you click this icon, the list of available fixes is displayed. Also, you can invoke the quick fix menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

 **Note:** The quick fixes are available only when validating an XSLT file with Saxon HE/PE/EE.

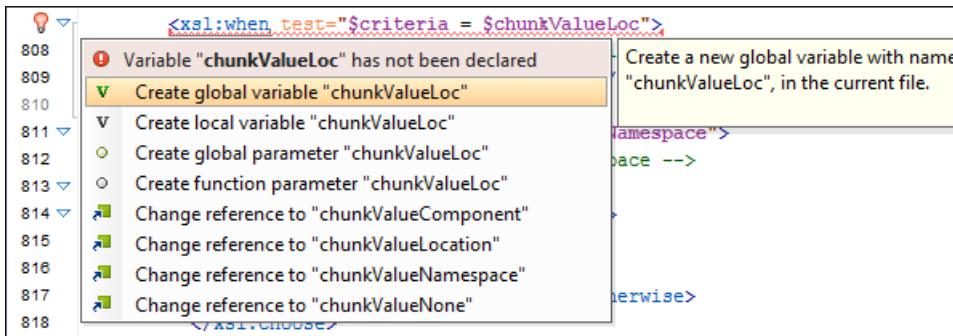


Figure 137: XSLT Variables and Parameters Quick Fix

Oxygen XML Editor provides quick fixes for the following cases:

- **Template does not exist**, when the template name referenced in a `call-template` element does not exist. The following fixes are available:
 - **Create template "templateName"** - creates a template and generates its corresponding parameters. The template name and parameter names and types are collected from the `call-template` element.
 - **Change reference to "newTemplateName"** - changes the name of the missing template referenced in the `call-template` element. The proposed new names are the existing templates with names similar with the missing one.
- **Variable/Parameter not declared**, when a parameter or variable reference cannot be found. The following fixes are available:
 - **Create global variable "varName"** - creates a global variable with the specified name in the current stylesheet. The new variable is added at the beginning of the stylesheet after the last global variable or parameter declaration.
 - **Create global parameter "paramName"** - creates a global parameter with the specified name in the current stylesheet. The new parameter is added at the beginning of the stylesheet after the last global parameter or variable declaration.
 - **Create local variable "varName"** - creates a local variable with the specified name before the current element.
 - **Create template parameter "paramName"** - creates a new parameter with the specified name in the current template. This fix is available if the error is located inside a template.
 - **Create function parameter "paramName"** - creates a new parameter with the specified name in the current function. This fix is available if the error is located inside a function.
 - **Change reference to "varName"** - changes the name of the referenced variable/parameter to an existing local or global variable/parameter, that has a similar name with the current one.
- **Parameter from a called template is not declared**, when a parameter referenced from a `call-template` element is not declared. The following fixes are available:
 - **Create parameter "paramName" in the template "templateName"** - creates a new parameter with the specified name in the referenced template.
 - **Change "paramName" parameter reference to "newParamName"** - changes the parameter reference from the `call-template` element to a parameter that is declared in the called template.
 - **Remove parameter "paramName" from call-template** - removes the parameter with the specified name from the `call-template` element.
- **No value supplied for required parameter**, when a required parameter from a template is not referenced in a `call-template` element. The following quick-fix is available:
 - **Add parameter "paramName" in call-template** - creates a new parameter with the specified name in `call-template` element.
- **Function "prefix:functionName()" has not been defined**, when a function declaration is not found. The following quick fixes are available:

- **Create function "prefix:functionName(param1, param2)"** - creates a new function with the specified signature, after the current top level element from stylesheet.
- **Change function to "newFunctionName(..)"** - changes the referenced function name to an already defined function. The proposed names are collected from functions with similar names and the same number of parameters.
- **Attribute-set "attrSetName" does not exist**, when the referenced attribute set does not exist. The following quick fixes are available:
 - **Create attribute-set "attrSetName"** - creates a new attribute set with the specified name, after the current top level element from stylesheet.
 - **Change reference to "attrSetName"** - changes the referenced attribute set to an already defined one.
- **Character-map "chacterMap" has not been defined**, when the referenced character map declaration is not found. The following quick fixes are available:
 - **Create character-map "characterMapName"** - creates a new character map with the specified name, after the current top level element from stylesheet.
 - **Change reference to "characterMapName"** - changes the referenced character map to an already defined one.

Linking Between Development and Authoring

The **Author** mode is available for the XSLT editor presenting the stylesheets in a nice visual rendering.

XSLT Unit Test (XSpec)

XSpec is a behavior driven development (BDD) framework for XSLT and XQuery. XSpec consists of a syntax for describing the behavior of your XSLT or XQuery code, and some code that enables you to test your code against those descriptions.

To create an XSLT Unit Test, go to **File > New > XSLT Unit Test**. You can also create an XSLT Unit Test from the contextual menu of an XSL file in the **Project** view. Oxygen XML Editor allows you to customize the XSpec document when you create it. In the customization dialog, you can enter the path to an XSL document or to a master XSL document.

To run an XSLT Unit Test, open the XSPEC file in an editor and click  **Apply Transformation Scenario(s)** on the main toolbar.



Note: The transformation scenario is defined in the XSPEC *document type*.

When you create an XSpec document based on an XSL document, Oxygen XML Editor uses information from the validation and transformation scenarios associated with the XSL file. From the transformation scenario Oxygen XML Editor uses extensions and properties of Saxon 9.6.0.5, improving the ANT scenario associated with the XSpec document.

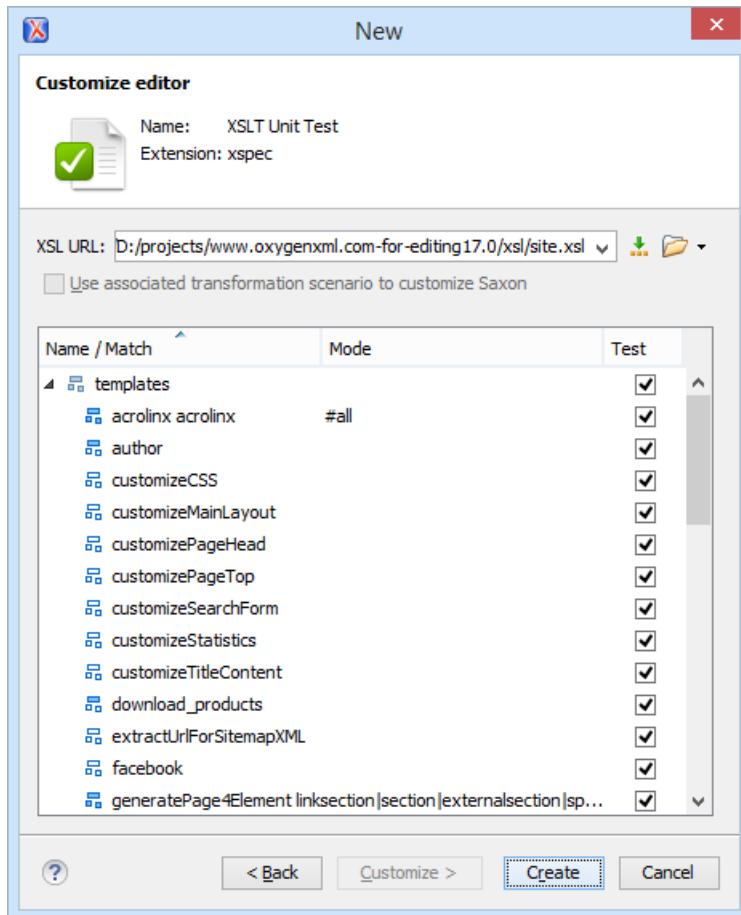


Figure 138: The New XSLT Unit Test wizard

An XSpec file contains one, or more test scenarios. You can test a stylesheet in one of the following ways:

- test an entire stylesheet;

Testing is performed in a certain context. You can define a context as follows:

- inline context - building the test based on a string;

```
<x:scenario label="when processing a para element">
  <x:context>
    <para>...</para>
  </x:context>
  ...
</x:scenario>
```

- based on an external file, or on a part of an external file extracted with an XPath expression.

```
<x:scenario label="when processing a para element">
  <x:context href="source/test.xml" select="/doc/body/p[1]" />
  ...
</x:scenario>
```

- test a function;

```
<x:scenario label="when capitalising a string">
  <x:call function="eg:capital-case">
    <x:param select="'an example string'" />
    <x:param select="true()" />
  </x:call>
  ...
</x:scenario>
```

- test a template with a name.

```
<x:call template="createTable">
  <x:param name="nodes">
    <value>A</value>
    <value>B</value>
  </x:param>
  <x:param name="cols" select="2" />
</x:call>
```

You are able to reference test files between each other, which allows you to define a suite of tests. For further details about test scenarios, go to <http://code.google.com/p/xspec/wiki/WritingScenarios>.

Editing Ant Build Files

This section explains the features of the Ant build files editor.

Validate Ant Build Files

Oxygen XML Editor performs the validation of Ant build files with the help of a built-in processor, which is largely based on the Apache Ant libraries. The path to these libraries can be configured in the [Ant Preferences](#) on page 1061 page. The validation processor accesses the [parameters set in the associated Ant transformation scenario](#) and uses them as Ant properties when validating the current build script.

To validate the Ant build file you are currently editing, select the **Validate** action from the **Validation** toolbar drop-down list or the **Document > Validate** menu.

Tip: To make a custom task available in the Ant validation engine, add a *JAR* file that contains the task implementation to the library directory of the built-in ANT distribution that comes bundled with Oxygen XML Editor (for example, `[oxygen Installation Directory]/tools/ant/lib` folder).

Custom Validation of Ant Build Files

If you need to validate an Ant build file with a validation engine that is different from the built-in engine, you can configure external engines as custom Ant validation engines in the Oxygen XML Editor preferences. After a custom validation engine is [properly configured](#), it can be applied on the current document by selecting it from the list of custom validation engines in the **Validation** toolbar drop-down list. The document is validated against the schema declared in the document.

Associate a Validation Scenario

You are able to validate Ant build files using a validation scenario. To create a validation scenario, select **Configure Validation Scenario(s)...** from the **Validation** toolbar drop-down list or the **Document > Validate** menu.

Editing Ant Build Files in the Master Files Context

Smaller interrelated modules that define a complex ant build file cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a template defined in a main build file is not visible when you edit an included or imported module. Oxygen XML Editor provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger Ant build structure.

You can set a main Ant build file either using the [master files support from the Project view](#), or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main build file. In this case, it considers the current module as the main build file.

The advantages of editing in the context of main file include:

- correct validation of a module in the context of a larger build structure;
- **Content Completion Assistant** displays all components valid in the current context;

- the **Outline** displays the components collected from the entire build file structure.

Syntax Highlight

To change the syntax highlighting colors for Ant, [open the Preferences dialog box](#) and go to **Editor > Colors**.

Content Completion in Ant Build Files

The items in the list of proposals offered by the **Content Completion Assistant** are context-sensitive.

The **Content Completion Assistant** proposes the following item types, defined in the current Ant build and in the imported and included builds:

- Property names.



Note: In addition to the user defined properties, the **Content Completion Assistant** offers the following values:

- The system properties set in the Java Virtual Machine.
- The built-in properties that Ant provides.

- Target names.
- Task and type reference IDs.



Tip: To make a custom task available in the **Content Completion Assistant**, add a *JAR* file that contains the task implementation to the library directory of the built-in ANT distribution that comes bundled with Oxygen XML Editor (for example, `[oxygen Installation Directory]/tools/ant/lib` folder).



Note: For Ant resources, the proposals are collected starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to [Defining Master Files at Project Level](#).

Ant Outline View

The Ant **Outline** view displays the list of all the components (properties, targets, extension points, task/type definitions and references) from both the edited Ant build file and its imported and included modules. For Ant resources, the **Outline** view collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to [Defining Master Files at Project Level](#). To enable the **Outline** view, go to **Window > Show View > Outline**.

The following actions are available in the **Settings** menu on the Outline view toolbar:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches;

➡ Selection update on caret move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret moves or the changes in the Ant editor. Selecting one of the components from the outline view also selects the corresponding item in the source document.



Show XML structure

Displays the XML document structure in a tree-like structure.

Show all components

Displays all components that were collected starting from the main file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/type

The build file components can be grouped by location and type.

Show components

Shows the define patterns collected from the current document.

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).

Configure displayed attributes

Displays the [XML Structured Outline preferences page](#).

The following contextual menu actions are available:

Append Child

Displays a list of elements that can be inserted as children of the current element.

Insert Before

Displays a list of elements that can be inserted as siblings of the current element, before the current element.

Insert After

Displays a list of elements that can be inserted as siblings of the current element, after the current element.

Edit Attributes

Displays an inline attribute editing window.

Toggle Comment

Comments/uncomments the currently selected element.

Search References [Ctrl Shift R \(Command Shift R on OS X\)](#)

Searches all references of the item found at current cursor position in the defined scope. See [Find References and Declarations of Ant Components](#) for more details.

Search References in...

Searches all references of the item found at current cursor position in the specified scope. See [Find References and Declarations of Ant Components](#) for more details.

Component Dependencies

Allows you to see the dependencies for the current selected component. See [Ant Component Dependencies View](#) for more details.

Rename Component in...

Renames the selected component. See [Ant Refactoring Actions](#) for more details.

You can search a component in the **Outline** view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type de component name in the text field, you can switch to the tree structure using:

- **down arrow key**
- **Tab key**
- **Shift-Tab** key combination

To switch from tree structure to the filter text field, you can use **Tab** and **Shift-Tab**.

 **Tip:** The search filter is case insensitive. The following wildcards are accepted:

- * - any string
- ? - any character
- , - patterns separator

If no wildcards are specified, the string to search is used as a partial match (like ***textToFind***).

On the Ant **Outline** view, there are available the following common contextual actions: **Edit Attributes**, **Cut**, **Copy**, **Delete**.

The **Outline** content and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

Oxygen XML Editor has a predefined order of the groups in the **Outline** view:

- For location, the names of the files are sorted alphabetically. The main file is the one you are editing and it is located at the top of the list.
- For type, the order is: properties, targets, references.



Note: When no grouping is available Oxygen XML Editor sorts the components depending on their order in the document. Oxygen XML Editor also takes into account the name of the file that the components are part of.

Find References and Declarations of Ant Components

The following actions are available for search operations related with references and declarations of Ant components:

- **Document > References > Search References** - Searches all references of the item found at current cursor position in the defined scope.
 - **Document > References > Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify after selecting a scope for the search operation.
 - **Document > References > Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope.
 - **Document > References > Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when defining a new scope.
 - **Document > References > Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
 - **Document > Schema > Show Definition** - Moves the cursor to the location of the definition of the current item.
- Note:** You can also use the **Ctrl Click (Command Click on OS X)** shortcut on a reference to display its definition.

Highlight Component Occurrences

When a component (for example *property* or *target*) is found at current cursor position, Oxygen XML Editor performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document.



Note: Oxygen XML Editor also supports occurrences highlight for type and task references.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is enabled by default. To configured it, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. If the automatic feature is disabled for a particular type of file, you can perform this search by going to **Search > Search Occurrences in File Ctrl Shift U (Command Shift U on OS X)** in the contextual menu. Matches are displayed in separate tabs of the **Results** view.

Ant Refactoring Actions

The following actions allow you to consistently rename a component in the entire Ant build file structure and are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

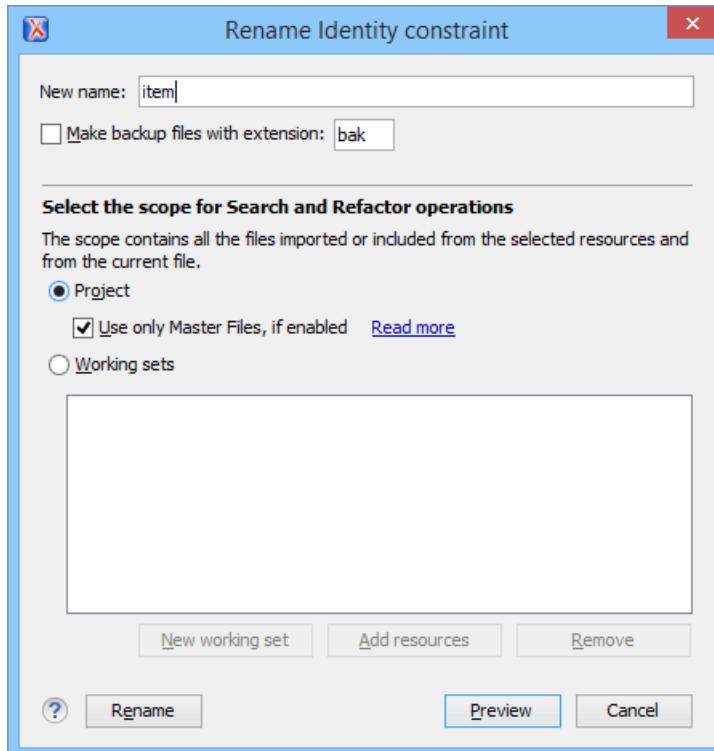


Figure 139: Rename Identity Constraint Dialog Box

Ant Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for an Ant build file. To open this view, go to **Window > Show View > Resource Hierarchy/Dependencies**.

If you want to see the hierarchy of a build file, select it in the project view and choose **Resource Hierarchy** from the contextual menu.

If you want to see the dependencies of a build file, select it in the project view and choose **Resource Dependencies** from the contextual menu.

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the  **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.



Tip: When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .

Ant Component Dependencies View

The Component Dependencies view allows you to see the dependencies for a selected Ant component. You can open the view from **Window > Show View > Component Dependencies** menu.

If you want to see the dependencies of an Ant component, select the desired component in the editor or **Outline** view and choose the **Component Dependencies** action from the contextual menu. The action is available for the following components: properties, targets (including extension points), and task or types references (those that have an id set).

In the Component Dependencies view there are available the following toolbar actions:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

History

Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

 **Tip:** If a component contains multiple references to another, these references are listed in a table displayed at the bottom of **Component Dependencies** view. When a recursive reference is encountered, it is marked with a special icon .

Ant Quick Assist Support

Quick Assist is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt 1 (Command Alt 1 on OS X)** on your keyboard.

The quick assist support offers direct access to the following actions:

Rename Component in...

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope...

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Ant Quick Fix Support

The Oxygen XML Editor Quick Fix support helps you resolve missing target reference errors that may occur when developing Ant build documents.

To activate the feature, when Oxygen XML Editor finds a validation error in an Ant build, place the caret in the highlighted area of text. If Oxygen XML Editor can provide a quick fix for that error, the  icon is displayed in the left side stripe. When you click this icon, the list of available fixes is displayed. Also, you can invoke the quick fix menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

Oxygen XML Editor provides the following quick fixes:

- **Create new target** - creates a new target with the specified name
- **Change reference to "targetName"** - corrects the reference to point to an already defined target
- **Remove target reference** - removes the erroneous reference

Editing XML Schemas

An XML Schema describes the structure of an XML document and is used to validate XML document instances against it, to check that the XML instances conform to the specified requirements. If an XML instance conforms to the schema then it is said to be valid, otherwise it is invalid.

Two editing modes are provided for working with XML Schema:

- The **Text** editing mode.
- The visual **Design** editing mode.



Note: Oxygen XML Editor offers support for both XML schema 1.0 and 1.1.

XML Schema Diagram Editing Mode

This section explains how to use the graphical diagram of a W3C XML Schema.

Introduction

XML Schemas enable document designers to specify the allowed structure and content of an XML document and to check if an XML document is valid.

Oxygen XML Editor provides a simple and expressive **Design** mode for editing XML Schemas. The schema diagram helps both the content authors who want to understand a schema and schema designers who develop complex schemas.

The diagram font can be increased using the usual Oxygen XML Editor shortcuts: [\(Ctrl \(Meta on Mac OS\) + "+"\)](#), [\(Ctrl \(Meta on Mac OS\)+"-"\)](#), [\(Ctrl \(Meta on Mac OS\) + 0\)](#) or [\(Ctrl \(Meta on Mac OS\) - mouse wheel\)](#). The whole diagram can also be zoomed with one of the predefined factors [available in the Schema preferences panel](#). The same zoom factor is applied for the print and save actions.

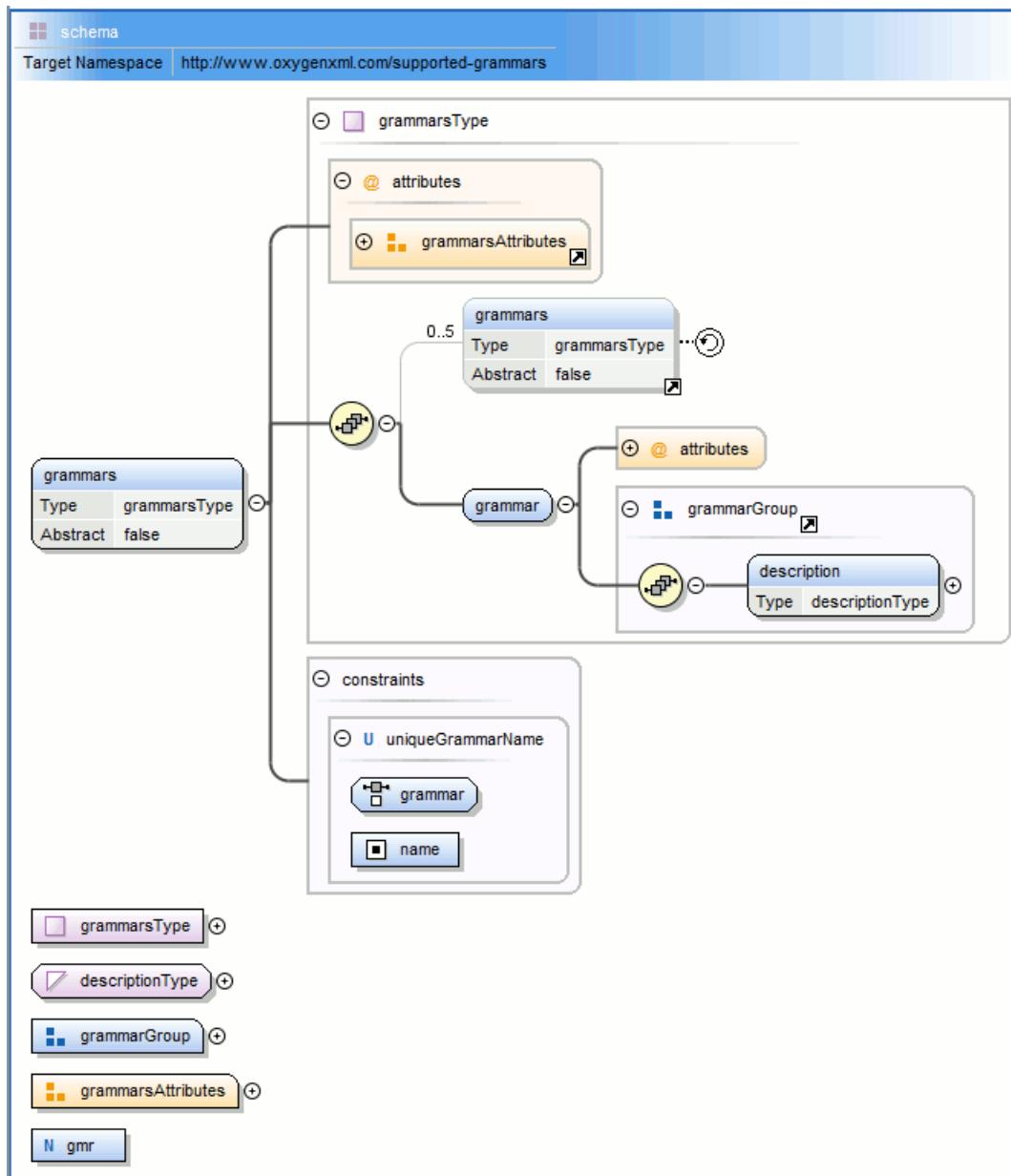


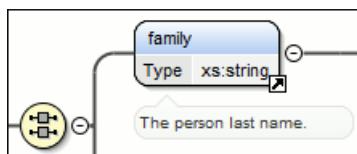
Figure 140: XML Schema Diagram

To watch our video demonstration about the basic aspects of designing an XML Schema using the new Schema Editor, go to http://oxygentools.com/demo/XML_Schema_Editing.html.

XML Schema Components

A schema diagram contains a series of interconnected components. To quickly identify the relation between two connected components, the connection is represented as:

- A thick line to identify a connection with a required component (in the following image, `family` is a required element).



- A thin line to identify a connection with an optional component (in the following image, `email` is an optional element).



The following topics explain in detail all available components and their symbols as they appear in an XML schema diagram.

xs:schema

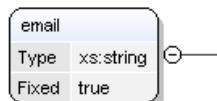
schema
Target Namespace | <http://www.oxygenxml.com/supported-grammars>

Defines the root element of a schema. A schema document contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-schema>.

By default it displays the `targetNamespace` property when rendered.

xs:schema properties

Property Name	Description	Possible Values
Target Namespace	The schema target namespace.	Any URI
Element Form Default	Determining whether local element declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Attribute Form Default	Determining whether local attribute declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Block Default	Default value of the <code>block</code> attribute of <code>xs:element</code> and <code>xs:complexType</code> .	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty].
Final Default	Default value of the <code>final</code> attribute of <code>xs:element</code> and <code>xs:complexType</code> .	#all, restriction, extension, restriction extension, [Empty].
Default Attributes	Specifies a set of attributes that apply to every complex Type in a schema document.	Any.
Xpath Default Namespace	The default namespace used when the XPath expression is evaluated.	##defaultNamespace, ##targetNamespace, ##local.
Version	Schema version	Any token.
ID	The schema id	Any ID.
Component	The edited component name.	Not editable property.
SystemID	The schema system id	Not editable property.

xs:element

Defines an element. An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-element>.

An element by default displays the following properties when rendered in the diagram: *default*, *fixed*, *abstract* and *type*. When referenced or declared locally, the element graphical representation also contains the value for the *minOccurs* and *maxOccurs* properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the element are drawn using dotted lines if the element is optional.

xs:element properties

Property Name	Description	Possible Values	Mentions
Name	The element name. Always required.	Any NCName for global or local elements, any QName for element references.	If missing, will be displayed as '[element]' in diagram.
Is Reference	When set, the local element is a reference to a global element.	true/false	Appears only for local elements.
Type	The element type.	All declared or built-in types. In addition, the following anonymous types are available: [ST-restriction], [ST-union], [ST-list], [CT-anonymous], [CT-extension SC], [CT-restriction SC], [CT-restriction CC], [CT-extension CC].	For all elements. For references, the value is set in the referenced element.
Base Type	The extended/restricted base type.	All declared or built-in types	For elements with complex type, with simple or complex content.
Mixed	Defines if the complex type content model will be mixed.	true/false	For elements with complex type.
Content	The content of the complex type.	simple/complex	For elements with complex type which extends/restricts a base type. It is automatically detected.
Content Mixed	Defines if the complex content model will be mixed.	true/false	For elements with complex type which has a complex content.

Property Name	Description	Possible Values	Mentions
Default	Default value of the element. A default value is automatically assigned to the element when no other value is specified.	Any string	The fixed and default attributes are mutually exclusive.
Fixed	A simple content element may be fixed to a specific value using this attribute. A fixed value is also automatically assigned to the element and you cannot specify another value.	Any string	The fixed and default attributes are mutually exclusive.
Min Occurs	Minimum number of occurrences of the element.	A numeric positive value. Default value is 1	Only for references/local elements
Max Occurs	Maximum number of occurrences of the element.	A numeric positive value. Default value is 1	Only for references/local elements
Substitution Group	Qualified name of the head of the substitution group to which this element belongs.	All declared elements. For XML Schema 1.1 reference elements	For global and reference elements this property supports multiple values.
Abstract	Controls whether the element may be used directly in instance XML documents. When set to true, the element may still be used to define content models, but it must be substituted through a substitution group in the instance document.	true/false	For global elements and element references
Form	Defines if the element is "qualified" (i.e., belongs to the target namespace) or "unqualified" (i.e., doesn't belong to any namespace).	unqualified/qualified	Only for local elements
Nillable	When this attribute is set to true, the element can be declared as nil using an <code>xsi:nil</code> attribute in the instance documents.	true/false	For global elements and element references

Property Name	Description	Possible Values	Mentions
Target Namespace	Specifies the target namespace for local element and attribute declarations. The namespace URI may be different from the schema target namespace. This property is available for local elements only.	Not editable property. For all elements.	
Block	Controls whether the element can be subject to a type or substitution group substitution. '#all' blocks any substitution, 'substitution' blocks any substitution through substitution groups and 'extension'/restriction' block any substitution (both through <code>xsi:type</code> and substitution groups) by elements or types, derived respectively by extension or restriction from the type of the element. Its default value is defined by the <code>blockDefault</code> attribute of the parent <code>xs:schema</code> .	#all, restriction, extension, substitution, and element references extension restriction, extension substitution, restriction substitution, restriction extension substitution	For global elements and element references
Final	Controls whether the element can be used as the head of a substitution group for elements whose types are derived by extension or restriction from the type of the element. Its default value is defined by the <code>finalDefault</code> attribute of the parent <code>xs:schema</code> .	#all, restriction, extension, restriction extension, [Empty]	For global elements and element references
ID	The component id.	Any id	For all elements.

Property Name	Description	Possible Values	Mentions
Component	The edited component name.	Not editable property. For all elements.	
Namespace	The component namespace.	Not editable property. For all elements.	
System ID	The component system id.	Not editable property. For all elements.	

xs:attribute

The manager ID.

Defines an attribute. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-attribute>.

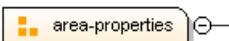
An attribute by default displays the following properties when rendered in the diagram: *default*, *fixed*, *use* and *type*. Connectors to the attribute are drawn using dotted lines if the attribute use is optional. The attribute name is stroked out if prohibited.

xs:attribute properties

Property Name	Description	Possible Value	Mentions
Name	Attribute name. Always required.	Any NCName for global/local attributes, all declared attributes' QName for references.	For all local or global attributes. If missing, will be displayed as '[attribute]' in the diagram.
Is Reference	When set, the local attribute is a reference.	true/false	For local attributes.
Type	Qualified name of a simple type.	All global simple types and built-in simple types. In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for creating anonymous simple types more easily.	For all attributes. For references, the type is set to the referenced attribute.
Default	Default value. When specified, an attribute is added by the schema processor (if it is missing from the instance XML document) and it is given this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the referenced attribute.
Fixed	When specified, the value of the attribute is fixed and must be equal to this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the referenced attribute.

Property Name	Description	Possible Value	Mentions
Use	Possible usage of the attribute. Marking an attribute "prohibited" is useful to exclude attributes during derivations by restriction.	optional, required, prohibited	For local attributes
Form	Specifies if the attribute is qualified (i.e., must have a namespace prefix in the instance XML document) or not. The default value for this attribute is specified by the attributeFormDefault attribute of the xs:schema document element.	unqualified/qualified	For local attributes.
Inheritable	Specifies if the attribute is inheritable. Inheritable attributes can be used by <alternative> element on descendant elements.	true/false	For all local or global attributes. The default value is false. This property is available for XML Schema 1.1.
Target Namespace	Specifies the target namespace for local attribute declarations. The namespace URI may be different from the schema target namespace.	Any URI	Setting a target namespace for local attribute is useful only when restricts attributes of a complex type that is declared in other schema with different target namespace. This property is available for XML Schema 1.1.
ID	The component id.	Any id	For all attributes.
Component	The edited component name.	Not editable property.	For all attributes.
Namespace	The component namespace.	Not editable property.	For all attributes.
System ID	The component system id.	Not editable property.	For all attributes.

xs:attributeGroup



The properties of an area.

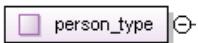
Defines an attribute group to be used in complex type definitions. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-attributeGroup>.

xs:attributeGroup properties

Property Name	Description	Possible Values	Mentions
Name	Attribute group name. Always required.	Any NCName for global attribute groups, all declared attribute groups for reference.	For all global or referenced attribute groups. If missing, will be displayed as '[attributeGroup]' in diagram.
ID	The component id.	Any id	For all attribute groups.

Property Name	Description	Possible Values	Mentions
Component	The edited component name. Not editable property.		For all attribute groups.
Namespace	The component namespace. Not editable property.		For all attribute groups.
System ID	The component system id. Not editable property.		For all attribute groups.

xs:complexType



Defines a top level complex type. Complex Type Definitions provide for: See more data at <http://www.w3.org/TR/xmlschema11-1/#element-complexType>.

- Constraining element information items by providing Attribute Declarations governing the appearance and content of attributes.
- Constraining element information item children to be empty, or to conform to a specified element-only or mixed content model, or else constraining the character information item children to conform to a specified simple type definition.
- Using the mechanisms of Type Definition Hierarchy to derive a complex type from another simple or complex type.
- Specifying post-schema-validation infoset contributions for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

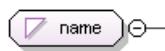
Tip: A complex type which is a base type to another type will be rendered with yellow background.

xs:complexType properties

Property Name	Description	Possible Values	Mentions
Name	The name of the complex type. Always required.	Any NCName	Only for global complex types. If missing, will be displayed as '[complexType]' in diagram.
Base Type Definition	The name of the extended/restricted types.	Any from the declared simple or complex types.	For complex types with simple or complex content.
Derivation Method	The derivation method.	restriction/ extension	Only when base type is set. If the base type is a simple type, the derivation method is always extension.
Content	The content of the complex type.	simple/ complex	For complex types which extend/restrict a base type. It is automatically detected.
Content Mixed	Specifies if the complex content model will be mixed.	true/false	For complex contents.
Mixed	Specifies if the complex type content model will be mixed.	true/false	For global and anonymous complex types.
Abstract	When set to true, this complex type cannot be used directly in the instance documents and needs to be substituted using an xsi:type attribute.	true/false	For global and anonymous complex types.

Property Name	Description	Possible Values	Mentions
Block	Controls whether a substitution (either through <code>xsi:type</code> or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the <code>blockDefault</code> attribute of <code>xs:schema</code> .	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Final	Controls whether the complex type can be further derived by extension or restriction to create new complex types.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Default Attributes Apply	The schema element can carry a <code>defaultAttributes</code> attribute, which identifies an attribute group. Each <code>complexType</code> defined in the schema document then automatically includes that attribute group, unless this is overridden by the <code>defaultAttributesApply</code> attribute on the <code>complexType</code> element.	true/false	This property is available only for XML Schema 1.1.
ID	The component id.	Any id	For all complex types.
Component	The edited component name.	Not editable property.	For all complex types.
Namespace	The component namespace.	Not editable property.	For all complex types.
System ID	The component system id.	Not editable property.	For all complex types.

xs:simpleType



The person name.

Defines a simple type. A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the normalized value of an attribute information item or of an element information item with no element children. Informally, it applies to the values of attributes and the text-only content of elements. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-simpleType>.

 **Tip:** A simple type which is a base type to another type will be rendered with yellow background.

xs:simpleType properties

Name	Description	Possible Values	Scope
Name	Simple type name. Always required.	Any NCName.	Only for global simple types. If missing, will be displayed as '[simpleType]' in diagram.
Derivation	The simple type category: restriction, list or union.	restriction, list or union	For all simple types.
Base Type	A simple type definition component. Required if derivation method is set to restriction.	All global simple types and built-in simple types. In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to restriction.
Item Type	A simple type definition component. Required if derivation method is set to list.	All global simple types and built-in simple types (from schema for schema). In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to list. Derivation by list is the process of transforming a simple datatype (named the item type) into a whitespace-separated list of values from this datatype. The item type can be defined inline by adding a simpleType definition as a child element of the list element, or by reference, using the itemType attribute (it is an error to use both).
Member Types	Category for grouping union members.	Not editable property.	For global and anonymous simple types with the derivation method set to union.
Member	A simple type definition component. Required if derivation method is set to union.	All global simple types and built-in simple types (from schema for schema). In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to union. Deriving a simple datatype by union merges the lexical spaces of several simple datatypes (called member types) to create a new simple datatype. The member types can be defined either by reference (through the memberTypes attribute) or embedded as simple datatype local definitions in the xs:union element. Both styles can be mixed.
Final	Blocks any further derivations of this datatype	#all, list, restriction, union, list restriction, list union,	Only for global simple types.

Name	Description	Possible Values	Scope
	(by list, union, derivation or all).	restriction union. In addition, [Empty] proposal is present for set empty string as value.	
ID	The component id.	Any id.	For all simple types
Component	The name of the edited component.	Not editable property.	Only for global and local simple types
Namespace	The component namespace.	Not editable property.	For global simple types.
System ID	The component system id.	Not editable property.	Not present for built-in simple types..

xs:alternative

The *type alternatives* mechanism allows you to specify type substitutions on an element declaration.



Note: xs:alternative is available for XML Schema 1.1.

htmlContentType
Test @type='html'

xs:alternative properties

Name	Description	Possible Values
Type	Specifies type substitutions for an element, depending on the value of the attributes.	All declared or built-in types. In addition, the following anonymous types are available: [ST-restriction], [ST-union], [ST-list], [CT-anonymous], [CT-extension SC], [CT-restriction SC], [CT-restriction CC], [CT-extension CC].
Test	Specifies an XPath expression. If the XPath condition is valid, the specified type is selected as the element type. The expressions allowed are limited to a subset of XPath 2.0. Only the attributes of the current element and inheritable attributes from ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.	An XPath expression.
XPath Default Namespace	The default namespace used when the XPath expression is evaluated.	##defaultNamespace, ##targetNamespace, ##local.
ID	Specifies the component ID.	Any ID.
Component	Specifies the type of XML schema component.	Not editable property.
System ID	Points to the document location of the schema.	Not editable property.

xs:group

Defines a group of elements to be used in complex type definitions. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-group>.

When referenced, the graphical representation also contains the value for the *minOccurs* and *maxOccurs* properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the group are drawn using dotted lines if the group is optional.

xs:group properties

Property Name	Description	Possible Values	Mentions
Name	The group name. Always required.	Any NCName for global groups, all declared groups for reference.	If missing, will be displayed as '[group]' in diagram.
Min Occurs	Minimum number of occurrences of the group.	A numeric positive value. Default value is 1.	Appears only for reference groups.
Max Occurs	Maximum number of occurrences of the group.	A numeric positive value. Default value is 1.	Appears only for reference groups.
ID	The component id.	Any id	For all groups.
Component	The edited component name.	Not editable property.	For all groups.
Namespace	The component namespace.	Not editable property	For all groups.
System ID	The component system id.	Not editable property.	For all groups.

xs:include

Adds multiple schemas with the same target namespace to a document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-include>.

xs:include properties

Property Name	Description	Possible Values
Schema Location	Included schema location.	Any URI
ID	Include ID.	Any ID
Component	The component name.	Not editable property.

xs:import

Adds multiple schemas with different target namespace to a document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-import>.

xs:import properties

Property Name	Description	Possible Values
Schema Location	Imported schema location	Any URI
Namespace	Imported schema namespace	Any URI
ID	Import ID	Any ID

Property Name	Description	Possible Values
Component	The component name	Not editable property.

xs:redefine

(+)  redefine: ./personal.xsd

Redefines simple and complex types, groups, and attribute groups from an external schema. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-redefine>.

xs:redefine properties

Property Name	Description	Possible Values
Schema Location	Redefine schema location.	Any URI
ID	Redefine ID	Any ID
Component	The component name.	Not editable property.

xs:override

(+)  override: invoice.xsd

The override construct allows replacements of old components with new ones without any constraint. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-override>.

xs:override properties

Property Name	Description	Possible Values
Schema Location	Redefine schema location.	Any URI
ID	Redefine ID	Any ID

xs:notation

(+) 

Describes the format of non-XML data within an XML document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-notation>.

xs:notation properties

Property Name	Description	Possible values	Mentions
Name	The notation name. Always required.	Any NCName.	If missing, will be displayed as '[notation]' in diagram.
System Identifier	The notation system identifier.	Any URI	Required if public identifier is absent, otherwise optional.
Public Identifier	The notation public identifier.	A Public ID value	Required if system identifier is absent, otherwise optional.
ID	The component id.	Any ID	For all notations.
Component	The edited component name.	Not editable property.	For all notations.
Namespace	The component namespace.	Not editable property.	For all notations.
System ID	The component system id.	Not editable property.	For all notations.

xs:sequence, xs:choice, xs:all**Figure 141: An xs:sequence in diagram**

xs : sequence specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-sequence>.

**Figure 142: An xs:choice in diagram**

xs : choice allows only one of the elements contained in the declaration to be present within the containing element. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-choice>.

**Figure 143: An xs:all in diagram**

xs : all specifies that the child elements can appear in any order. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-all>.

The compositor graphical representation also contains the value for the minOccurs and maxOccurs properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the compositor are drawn using dotted lines if the compositor is optional.

xs:sequence, xs:choice, xs:all properties

Property Name	Description	Possible Values	Mentions
Compositor	Compositor type.	sequence, choice, all.	'all' is only available as a child of a group or complex type.
Min Occurs	Minimum occurrences of compositor.	A numeric positive value. Default is 1.	The property is not present if compositor is 'all' and is child of a group.
Max Occurs	Maximum occurrences of compositor.	A numeric positive value. Default is 1.	The property is not present if compositor is 'all' and is child of a group.
ID	The component id.	Any ID	For all compostors.
Component	The edited component name.	Not editable property.	For all compostors.
System ID	The component system id.	Not editable property.	For all compostors.

xs:any

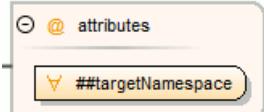
Enables the author to extend the XML document with elements not specified by the schema. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-any>.

The graphical representation also contains the value for the minOccurs and maxOccurs properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the wildcard are drawn using dotted lines if the wildcard is optional.

xs:any properties

Property Name	Description	Possible Values
Namespace	The list of allowed namespaces. The namespace attribute expects a list of namespace URIs. In this list, two values have a specific meaning: '#targetNamespace' stands for the target namespace, and '#local' stands for local attributes (without namespaces).	##any, ##other, ##targetNamespace, ##local or anyURI
notNamespace	Specifies the namespace that extension elements or attributes cannot come from.	##local, ##targetNamespace
notQName	Specifies an element or attribute that is not allowed.	##defined
Process Contents	Type of validation required on the elements allowed for this wildcard.	skip, lax, strict
Min Occurs	Minimum occurrences of any	A numeric positive value. Default is 1.
Max Occurs	Maximum occurrences of any	A numeric positive value. Default is 1.
ID	The component id.	Any ID.
Component	The name of the edited component.	Not editable property.
System ID	The component system id.	Not editable property.

xs:anyAttribute



Enables the author to extend the XML document with attributes not specified by the schema. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-anyAttribute>.

xs:anyAttribute properties

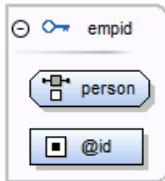
Property Name	Description	Possible Value
Namespace	The list of allowed namespaces. The namespace attribute expects a list of namespace URIs. In this list, two values have a specific meaning: '#targetNamespace' stands for the target namespace, and '#local' stands for local attributes (without namespaces).	##any, ##other, ##targetNamespace, ##local or anyURI
Process Contents	Type of validation required on the elements allowed for this wildcard.	skip, lax, strict
ID	The component id.	Any ID.
Component	The name of the edited component.	Not editable property.
System ID	The component system id.	Not editable property.

xs:unique

Defines that an element or an attribute value must be unique within the scope. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-unique>.

xs:unique properties

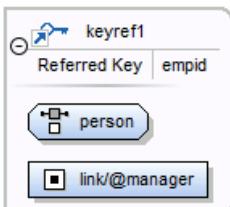
Property Name	Description	Possible Values
Name	The unique name. Always required.	Any NCName.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
Namespace	The component namespace.	Not editable property.
System ID	The component system id.	Not editable property.

xs:key

Specifies an attribute or element value as a key (unique, non-nullable and always present) within the containing element in an instance document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-key>.

xs:key properties

Property Name	Description	Possible Value
Name	The key name. Always required.	Any NCName.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
Namespace	The component namespace.	Not editable property.
System ID	The component system id.	Not editable property.

xs:keyRef

Specifies that an attribute or element value corresponds to that of the specified key or unique element. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-keyref>.

A keyref by default displays the *Referenced Key* property when rendered.

xs:keyRef properties

Property Name	Description	Possible Values
Name	The keyref name. Always required.	Any NCName.
Referenced Key	The name of referenced key.	any declared element constraints.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
Namespace	The component namespace.	Not editable property.
System ID	The component system id.	Not editable property.

xs:selector



Specifies an XPath expression that selects a set of elements for an identity constraint. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-selector>.

xs:selector properties

Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the element on which the constraint applies.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not editable property.

xs:field



Specifies an XPath expression that specifies the value used to define an identity constraint. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-field>.

xs:field properties

Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the field(s) composing the key, key reference, or unique constraint.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not editable property.

xs:assert

Assertions provide a flexible way to control the occurrence and values of elements and attributes available in an XML Schema.



Note: xs:assert is available for XML Schema 1.1.

a < b @minPrice le @maxPrice

xs:assert properties

Property Name	Description	Possible Values
Test	Specifies an XPath expression. If the XPath condition is valid, the specified type is selected as the element type. The expressions allowed are limited to a subset of XPath 2.0. Only the attributes of the current element and inheritable attributes from ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.	An XPath expression.
XPath Default Namespace	The default namespace used when the XPath expression is evaluated.	##defaultNamespace, ##targetNamespace, ##local.
ID	Specifies the component ID.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not editable property.

xs:openContent



The openContent element enables instance documents to contain extension elements interleaved among the elements declared by the schema. You can declare open content for your elements at one place - within the complexType definition, or at the schema level.

For further details about the openContent component, go to <http://www.w3.org/TR/xmlschema11-1/#element-openContent>.

xs:openContent properties

Property Name	Description	Possible Value
Mode	Specifies where the extension elements can be inserted.	The value can be: "interleave", "suffix" or "none". The default value is "interleave".
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not editable property.

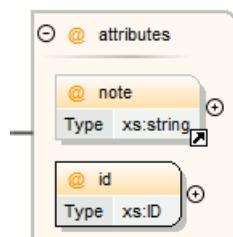


Note: This component is available for XML Schema 1.1 only. To change the version of the XML Schema, [open the Preferences dialog box](#) and go to **XML > XML Parser > XML Schema**.

Constructs Used to Group Schema Components

This section explains the components that can be used for grouping other schema components:

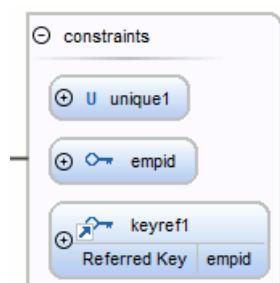
- [Attributes](#)
- [Constraints](#)
- [Substitutions](#)

Attributes

Groups all attributes and attribute groups belonging to a complex type.

Attributes properties

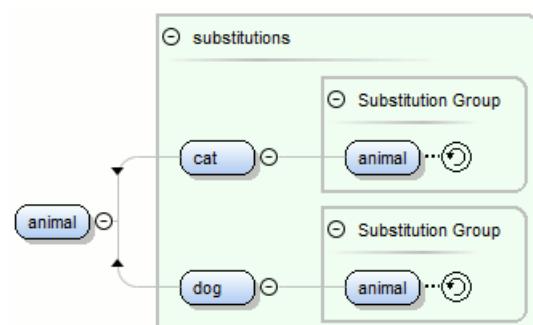
Property Name	Description	Possible Values
Component	The element for which the attributes are displayed.	Not editable property.
System ID	The component system id.	Not editable property.

Constraints

Groups all constraints ([xs:key](#), [xs:keyRef](#) or [xs:unique](#)) belonging to an element.

Attributes properties

Property Name	Description	Possible Values
Component	The element for which the constraints are displayed.	Not editable property.
System ID	The component system id.	Not editable property.

Substitutions

Groups all elements which can substitute the current element.

Attributes properties

Property Name	Description	Possible Values
Component	The element for which the substitutions are displayed.	Not editable property.
System ID	The component system id.	Not editable property.

Navigation in the Schema Diagram

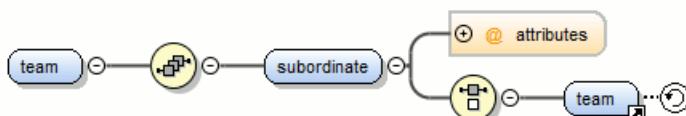
The following editing and navigation features work for all types of schema components:

- Move/reference components in the diagram using drag-and-drop actions.
- Select consecutive components on the diagram (components from the same level) using the *Shift* key. You can also make discontinuous selections in the schema diagram using the **Ctrl (Meta on Mac OS)** key. To deselect one of the components, use **Ctrl Click (Command Click on OS X)**.
- Use the arrow keys to navigate the diagram vertically and horizontally.
- Use *Home/End* keys to navigate to the first/last component from the same level. Use **Ctrl Home (Command Home on OS X)** key combination to go to the diagram root and **Ctrl End (Command End on OS X)** to go to the last child of the selected component.
- You can easily go back to a previously visited component while moving from left to right. The path will be preserved only if you use the left arrow key or right arrow key. For example, if the current selection is on the second attribute from an attribute group and you press the left arrow key to navigate to the attribute group, when you press the right arrow key, then the selection will be moved to the second attribute.
- Go back and forward between components viewed or edited in the diagram by selecting them in the **Outline** view:
 - Back** (go to previous schema component).
 - Forward** (go to next schema component).
 - Go to Last Modification** (go to last modified schema component).
- Copy, reference, or move global components, attributes, and identity constraints to a different position and from one schema to another using the **Cut/Copy** and **Paste/Paste as Reference** actions.
- Go to the definition of an element or attribute with the **Show Definition** action.
- Search in the diagram using the **Find/Replace dialog** or the **Quick find toolbar**. You can find/replace components only in the current file scope.
- You can expand and see the contents of the imports/includes/redefines in the diagram. In order to edit components from other schemas the schema for each component will be opened as a separate file in Oxygen XML Editor.



Tip: If an XML Schema referenced by the current opened schema was modified on disk, the change will be detected and you will be asked to refresh the current schema contents.

- Recursive references are marked with a *recurse symbol*: Click this symbol to navigate between the element declaration and its reference.



Schema Validation

Validation for the **Design** mode is seamlessly integrated in the Oxygen XML Editor **XML documents validation** capability.

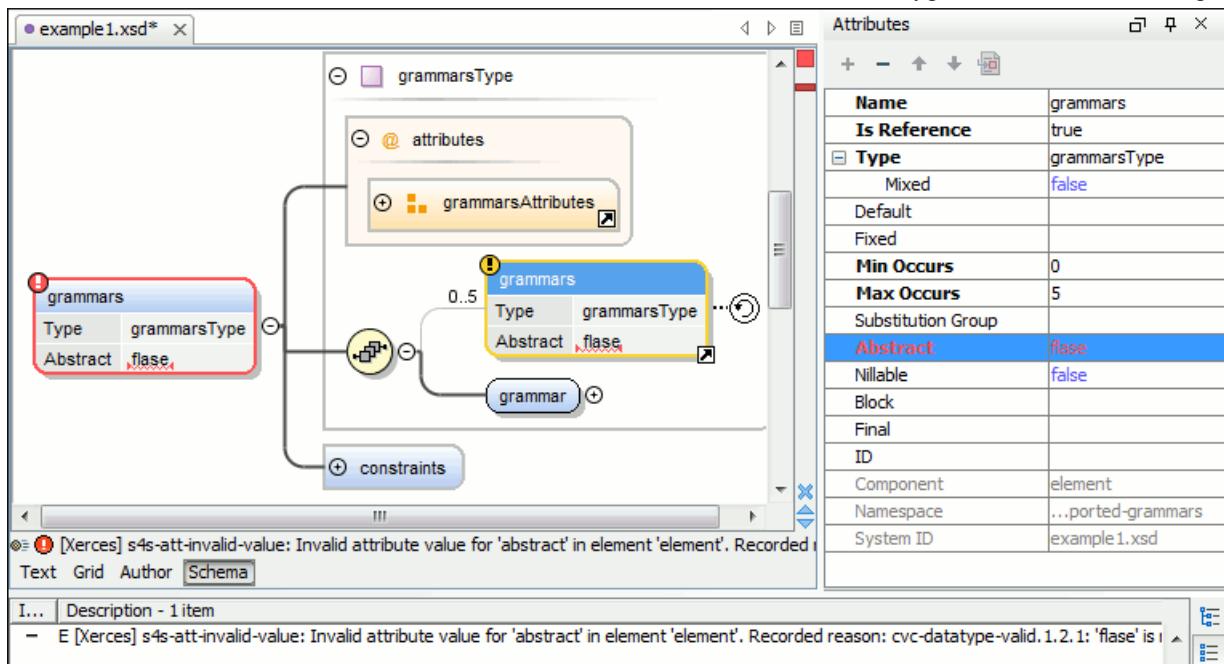


Figure 144: XML Schema Validation

A schema validation error is presented by highlighting the invalid component:

- In the *Attributes View*.
- In the diagram by surrounding the component that has the error with a red border.
- A marker on the errors stripe at the right of the diagram view.
- A status label with a red icon (!) below the diagram view.

Invalid facets for a component are highlighted in the *Facets View*.

Components with invalid properties are rendered with a red border. This is a default color, but you can customize it in the *Document checking user preferences*. When hovering an invalid component, the tooltip will present the validation errors associated with that component.

When editing a value which is supposed to be a qualified or unqualified XML name, the application provides automatic validation of the entered value. This proves to be very useful in avoiding setting invalid XML names for the given property.

If you validate the entire schema using the **Validate** action from the **Document > Validate** menu or from the **Validation** toolbar drop-down list, all validation errors will be presented in the **Errors** tab. To resolve an error, just click on it (or double click for errors located in other schemas) and the corresponding schema component will be displayed as the diagram root so that you can easily correct the error.

Important: If the schema imports only the namespace of other schema without specifying the schema location and a *catalog is set-up* that maps the namespace to a certain location both the validation and the diagram will correctly identify the imported schema.

Tip: If the validation action finds that the schema contains unresolved references, the application will suggest the use of validation scenarios, but only if the current edited schema is a XML Schema module.

Schema Editing Actions

You can edit an XML schema using drag and drop operations or contextual menu actions.

Drag and drop is the easiest way to move the existing components to other locations in an XML schema. For example, you can quickly insert an element reference in the diagram with a drag and drop from the **Outline** view to a compositor in the diagram. Also, the components order in an `xs:sequence` can be easily changed using drag and drop.

If this property has not been set, you can easily set the attribute/element type by dragging over it a simple type or complex type from the diagram. If the type property for a simple type or complex type is not already set, you can set it by dragging over it a simple or complex type.

Depending on the drop area, different actions are available:

- **move** - Context dependent, the selected component is moved to the destination.
- **reference** - Context dependent, the selected component is referenced from the parent.
- **copy** - If (**Ctrl (Meta on Mac OS)**) key is pressed, a copy of the selected component is inserted to the destination.

Visual clues about the operation type are indicated by the mouse pointer shape:

-  - When moving a component.
-  - When referencing a component.
-  - When copying a component.

You can edit some schema components directly in the diagram. For these components, you can edit the name and the additional properties presented in the diagram by double clicking the value you want to edit. If you want to edit the name of a selected component, you can also press (**Enter**). The list of properties which can be displayed for each component can be customized *in the Preferences*.

When editing references, you can choose from a list of available components. Components from an imported schema for which the target namespace does not have an associated prefix is displayed in the list as `componentName#targetNamespace`. If the reference is from a target namespace which was not yet mapped, you are prompted to add prefix mappings for the inserted component namespace in the current edited schema.

You can also change the compositor by double-clicking it and choose the compositor you want from the proposals list.

There are some components that cannot be edited directly in the diagram: imports, includes, redefines. The editing action can be performed if you double-click or press (**Enter**) on an import/include/redefine component. An edit dialog is displayed, allowing you to customize the directives.

Contextual Menu Actions in the Design Mode

The contextual menu of the **Design** mode offers the following edit actions:

Show Definition (Ctrl Shift ENTER (Command Shift ENTER on OS X))

Shows the definition for the current selected component. For references, this action is available by clicking the arrow displayed in its bottom right corner.

Open Schema (Ctrl Shift ENTER (Command Shift ENTER on OS X))

Opens the selected schema. This action is available for `xsd:import`, `xsd:include` and `xsd:redefine` elements. If the file you try to open does not exist, a warning message is displayed and you have the possibility to create the file.

Edit Attributes... ()

Allows you to edit the attributes of the selected component in a dialog box that presents the same attributes as in the *Attributes View* and the *Facets View*. The actions that can be performed on attributes in this dialog box are the same actions presented in the two views.

Append child

Offers a list of valid components to append depending on the context. For example to a complex type you can append a compositor, a group, attributes or identity constraints (`unique`, `key`, `keyref`). You can set a name for a named component after it was added in the diagram.

Insert before

Inserts before the selected component in the schema. The list of components that can be inserted depends on the context. For example, before an `xsd:import` you can insert an `xsd:import`, `xsd:include` or `xsd:redefine`. You can set a name for a named component after it was added in the diagram.

Insert after

Inserts a component after the selected component on the schema. The list of components that can be inserted depends on the context. You can set a name for a named component after it was added in the diagram.

New global

Inserts a global component in the schema diagram. This action does not depend on the current context. If you choose to insert an import you have to specify the URL of the imported file, the target namespace and the import ID. The same information, excluding the target namespace, is requested for an `xsd:include` or `xsd:redefine` element.



Note: If the imported file has declared a target namespace, the field **Namespace** is completed automatically.

Edit Schema Namespaces...

When performed on the schema root, it allows you to edit the schema target namespace and namespace mappings. You can also invoke the action by double-clicking the target namespace property from **Attributes** view for the schema or by double-clicking the schema component.

Edit Annotations...

Allows you to edit the annotation for the selected schema component in the **Edit Annotations** dialog box. You can perform the following operations in the dialog box:

- **Edit all appinfo/documentation items for a specific annotation** - all `appinfo/documentation` items for a specific annotation are presented in a table and can be easily edited. Information about an annotation item includes: type (`documentation/appinfo`), content, source (optional, specify the source of the `documentation/appinfo` element) and `xml:lang`. The content of a `documentation/appinfo` item can be edited in the **Content** area below the table.
- **Insert/Insert before/Remove documentation/appinfo.** The **Add** button allows you to insert a new annotation item (`documentation/appinfo`). You can add a new item before the item selected in table by pressing the **Insert Before** button. Also, you can delete the selected item using the **Remove** button.
- **Move items up/down** - to do this use the **Move up** and **Move down** buttons.
- **Insert/Insert before/Remove annotation** - available for components that allow multiple annotations like schemas or redefines.
- **Specify an ID for the component annotation.** the ID is optional.

Annotations are rendered by default under the graphical representation of the component. When you have a reference to a component with annotations, these annotations are presented in the diagram also below the reference component. The **Edit Annotations** action invoked from the contextual menu edit the annotations for the reference. If the reference component does not have annotations, you can edit the annotations of the referenced component by double-clicking the annotations area. Otherwise you can edit the referenced component annotations only if you go to the definition of the component.



Note: For imported/included components which do not belong to the currently edited schema, the **Edit Annotations** dialog box presents the annotation as read-only. To edit its annotation, open the schema where the component is defined.

Extract Global Element

Action available for local elements. A local element is made global and is replaced with a reference to the global element. The local element properties that are also valid for the global element declaration are kept.

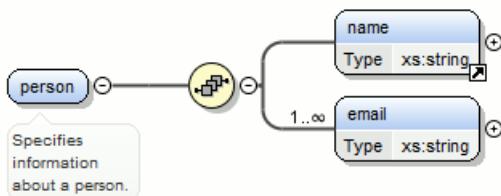
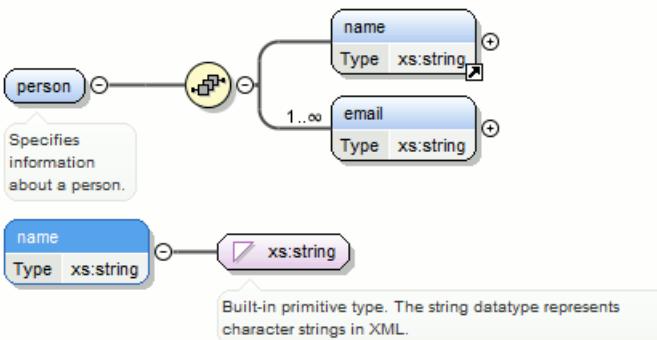


Figure 145: Extracting a Global Element

If you execute **Extract Global Element** on element **name**, the result is:



Extract Global Attribute

Action available for local attributes. A local attribute is made global and replaced with a reference to the global attribute. The properties of local attribute that are also valid in the global attribute declaration are kept.

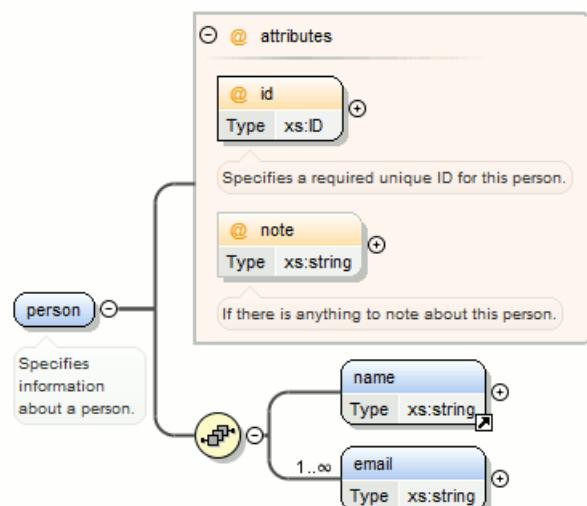
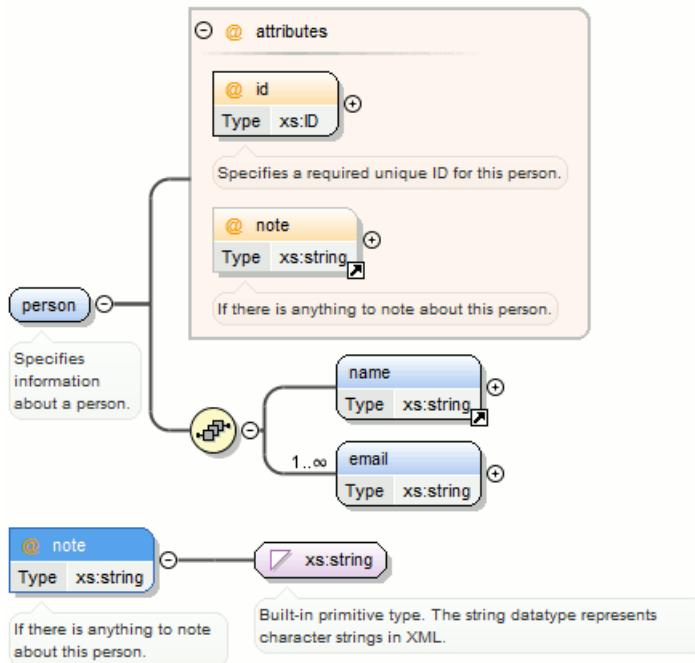


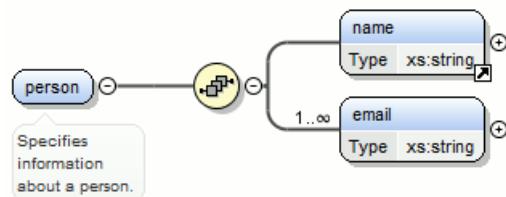
Figure 146: Extracting a Global Attribute

If you execute **Extract Global Attribute** on attribute **note** the result is:

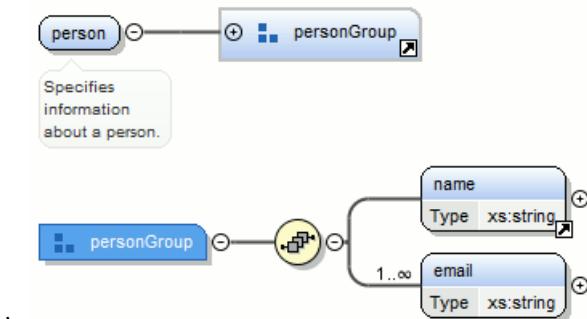


Extract Global Group

Action available for compositors (sequence, choice, all). This action extracts a global group and makes a reference to it. The action is enabled only if the parent of the compositor is not a group.



If you execute **Extract Global Group** on the sequence element, the **Extract Global Component** dialog box is shown and you can choose a name for the group. If you type `personGroup`, the result



is:

Figure 147: Extracting a Global Group

Extract Global Type

Action used to extract an anonymous simple type or an anonymous complex type as global. For anonymous complex types, the action is available on the parent element.

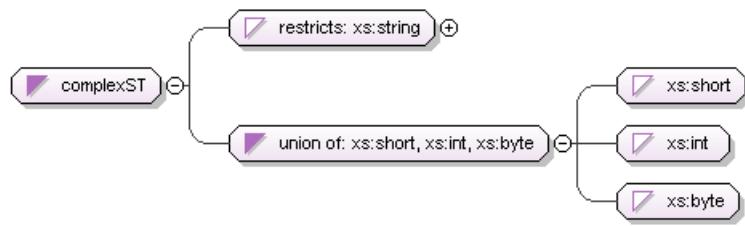


Figure 148: Extracting a Global Simple Type

If you use the action on the union component and choose `numericST` for the new global simple type name, the result is:

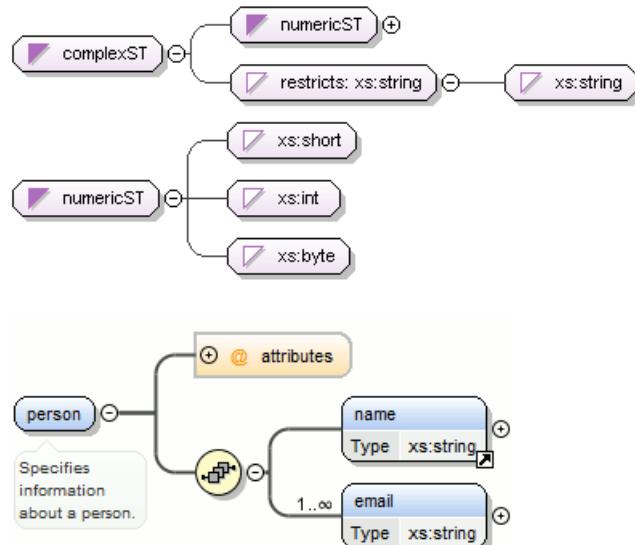
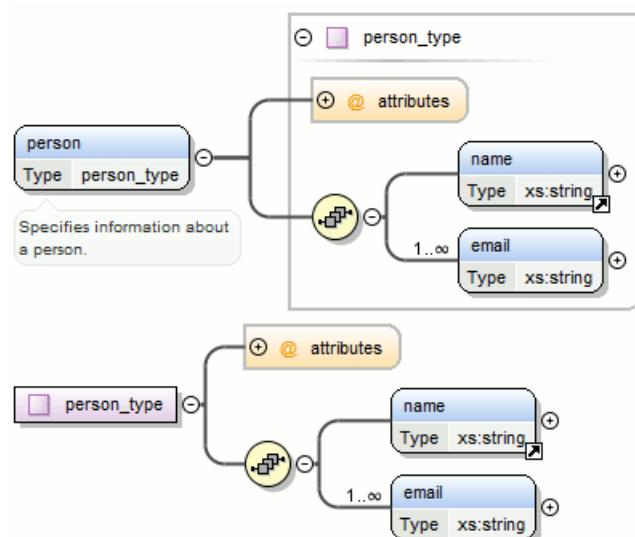


Figure 149: Extracting a Global Complex Type

If you execute the action on element `person` and choose `person_type` for the new complex type name, the result is:



Rename Component in...

Rename the selected component.

 **Cut Ctrl X (Command X on OS X)**

Cut the selected component(s).

 **Copy Ctrl C (Command C on OS X)**

Copy the selected component(s).

 **Copy XPath**

This action copies an XPath expression that identifies the selected element or attribute in an instance XML document of the edited schema and places it in the clipboard.

 **Paste Ctrl V (Command V on OS X)**

Paste the component(s) from the clipboard as children of the selected component.

Paste as Reference

Create references to the copied component(s). If not possible a warning message is displayed.

Remove (Delete)

Remove the selected component(s).

Override component

Copies the overridden component in the current XML Schema. This option is available for *xs:override* components.

Redefine component

The referenced component is added in the current XML Schema. This option is available for *xs:redesign* components.

Optional

Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The *minOccurs* property is set to 0 and the *use* property for attributes is set to optional.

Unbounded

Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The *maxOccurs* property is set to unbounded and the *use* property for attributes is set to required.

Search

Can be performed on local elements or attributes. This action makes a reference to a global element or attribute.

 **Search References**

Searches all references of the item found at current cursor position in the defined scope if any.

Search References in...

Searches all references of the item found at current cursor position in the specified scope.

Search Occurrences in File

Searches all occurrences of the item found at current cursor position in the current file.

 **Component Dependencies**

Allows you to see the dependencies for the current selected component.

Resource Hierarchy

Allows you to see the hierarchy for the current selected resource.

Flatten Schema

Recursively adds the components of included Schema files to the main one. It also flattens every imported XML Schema from the hierarchy.

Resource Dependencies

Allows you to see the dependencies for the current selected resource.

 **Expand all**

Expands recursively all sub-components of the selected component.

 **Collapse all**

Collapses recursively all sub-components of the selected component.

 **Save as Image...**

Save the diagram as image, in JPEG, BMP, SVG or PNG format.

 **Generate Sample XML Files**

Generate XML files using the current opened schema. The selected component is the XML document root. See more in the [Generate Sample XML Files](#) section.

 **Options...**

Show the [Schema preferences panel](#).

Schema Outline View

The **Outline** view presents all the global components grouped by their location, namespace, or type. If hidden, you can open it from **Window > Show View > Outline**.

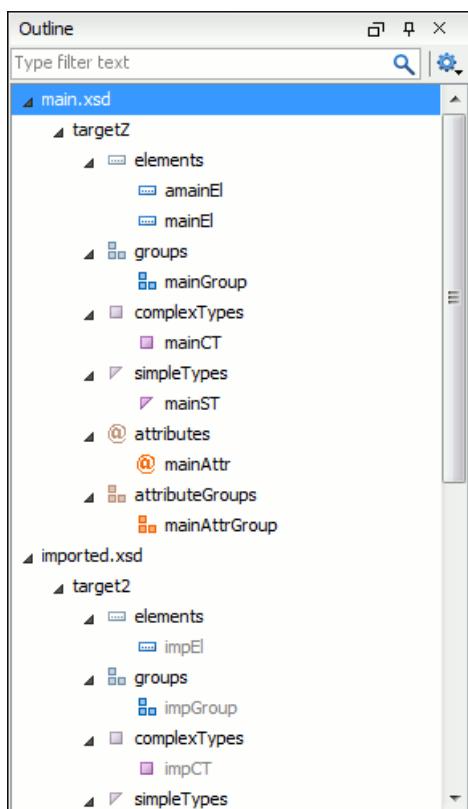


Figure 150: The Outline View for XML Schema

The **Outline** view provides the following options:

 **Selection update on caret move**

Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram is also selected in the **Outline** view.

 **Sort**

Allows you to sort alphabetically the schema components.

Show all components

Displays all components that were collected starting from the main files. Components that are not referable from the current file are marked with an orange underline. To reference them, add an import directive with the componentNS namespace.

Show referable components

Displays all components (collected starting from the main files) that can be referenced from the current file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

These three operations allow you to group the components by location, namespace, or type. When grouping by namespace, the main schema target namespace is the first presented in the **Outline** view.

The following contextual menu actions are available:

Remove (Delete)

Removes the selected item from the diagram.

Search References (Ctrl (Meta on Mac OS)+Shift+R)

Searches all references of the item found at current cursor position in the defined scope, if any.

Search References in...

Searches all references of the item found at current cursor position in the specified scope.

Component Dependencies (Ctrl (Meta on Mac OS)+Shift+F4)

Allows you to see the dependencies for the current selected component.

Resource Hierarchy (F4)

Allows you to see the hierarchy for the current selected resource.

Resource Dependencies (Shift + F4)

Allows you to see the dependencies for the current selected resource.

Rename Component in...

Renames the selected component.

Generate Sample XML Files...

Generate XML files using the current opened schema. The selected component is the XML document root.

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

 **Tip:** The search filter is case insensitive. The following wildcards are accepted:

- * - any string
- ? - any character
- , - patterns separator

If no wildcards are specified, the string to search will be searched as a partial match (like *textToFind*).

The **Outline** content and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

The Attributes View

The **Attributes** view presents the properties for the selected component in the schema diagram. If hidden, you can open it from **Window > Show View > Attributes**.

Attributes	
	
Name	family
Type	[ST - union]
Member Types	
Member	xs:string
Default	
Fixed	
Substitution Group	
Abstract	false
Nillable	false
Block	
Final	
ID	
Component	element
Namespace	
System ID	personal.xsd

Figure 151: The Attributes View

The default value of a property is presented in the **Attributes** view with blue foreground. The properties that can't be edited are rendered with gray foreground. A non-editable category which contains at least one child is rendered with bold. Bold properties are properties with values set explicitly to them.

Properties for components which do not belong to the current edited schema are read-only but if you double-click them you can choose to open the corresponding schema and edit them.

You can edit a property by double-clicking on or by pressing Enter. For most properties you can choose valid values from a list or you can specify another value. If a property has an invalid value or a warning, it will be highlighted in the table with the corresponding foreground color. By default properties with errors are highlighted with red and the properties with warnings are highlighted with yellow. You can customize these colors from the [Document checking user preferences](#).

For imports, includes and redefines, the properties are not edited directly in the **Attributes** view. A dialog will be shown allowing you to specify properties for them.

The schema namespace mappings are not presented in **Attributes** view. You can view/edit these by choosing **Edit Schema Namespaces** from the contextual menu on the schema root. See more in the [Edit Schema Namespaces](#) section.

The **Attributes** view has five actions available on the toolbar and also on the contextual menu:

 **Add**

Allows you to add a new member type to an union's member types category.

 **Remove**

Allows you to remove the value of a property.

 **Move Up**

Allows you to move up the current member to an union's member types category.

 **Move Down**

Allows you to move down the current member to an union's member types category.

 **Copy**

Copy the attribute value.

 **Show Definition**[Ctrl \(Meta on MAC OS\) + Click](#)

Shows the definition for the selected type.

Show Facets

Allows you to edit the facets for a simple type.

The Facets View

The **Facets** view presents the facets for the selected component, if available. If hidden, you can open it from **Window > Show View > Facets**.

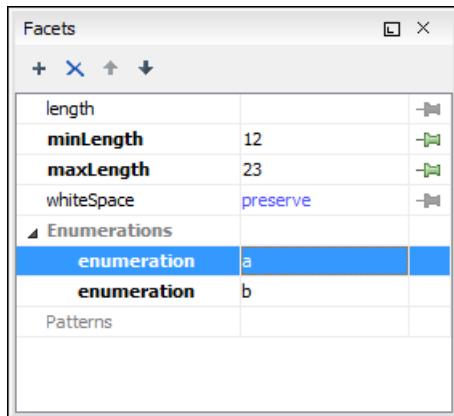


Figure 152: The Facets View

The default value of a facet is rendered in the **Facets** view with a blue color. The facets that can't be edited are rendered with a gray color. The grouping categories (for example: **Enumerations** and **Patterns**) are not editable. If these categories contain at least one child they are rendered with bold. Bold facets are facets with values set explicitly to them.

Important: Usually inherited facets are presented as default in the **Facets** view but if patterns are inherited from a base type and also specified in the current simple type only the current specified patterns will be presented. You can see the effective pattern value obtained by combining the inherited and the specified patterns as a tooltip on the **Patterns** category.

Facets for components which do not belong to the current edited schema are read-only but if you double-click them you can choose to open the corresponding schema and edit them.

You can edit a facet by double-clicking on it or by pressing Enter, when that facet is selected. For some facets you can choose valid values from a list or you can specify another value. If a facet has an invalid value or a warning, it will be highlighted in the table with the corresponding foreground color. By default facets with errors are presented with red and the facets with warnings with yellow. You can customize the error colors from the [Document Checking user preferences](#).

The **Facets** view has four toolbar actions available also on the contextual menu:

Add

Allows you to add a new enumeration or a new pattern.

Remove

Allows you to remove the value of a facet.

Move Up

Allows you to move up the current enumeration/pattern in **Enumerations/Patterns** category.

Move Down

Allows you to move down the current enumeration/pattern in **Enumerations/Patterns** category.

Copy

Copy the attribute value.

Open in Regular Expressions Builder

Allows you to open the pattern in the [XML Schema Regular Expressions Builder](#).

Facets can be fixed to prevent a derivation from modifying its value. To fix a facet value just press the **Pin** button.

Editing Patterns

You are able to edit regular expressions either by hand, or using the **Open in Regular Expression Builder** action from the contextual menu. This action offers a full-fledged *XML Schema Regular Expression builder* that guides through testing and constructing the pattern.

The Palette View

The **Palette** view is designed to offer quick access to XML Schema components and to improve the usability of the XML Schema diagram builder. You can use the **Palette** to drag and drop components in the **Design** mode. The components offered in the **Palette** view depend on the XML schema version set in the [XML Schema preferences page](#).

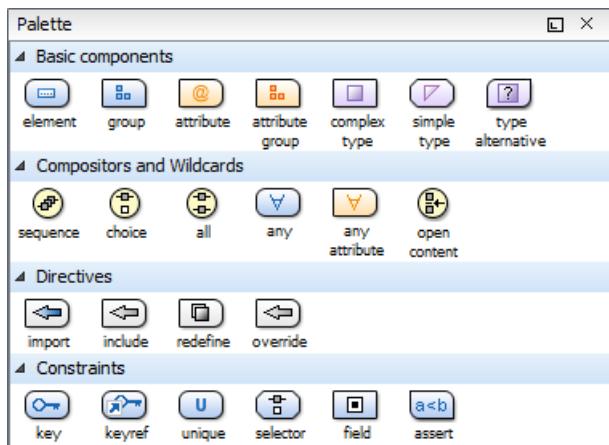


Figure 153: Palette View

Components are organized functionally into 4 collapsible categories:

- Basic components: *elements, group, attribute, attribute group, complex type, simple type, type alternative*.
- Compositors and Wildcards: *sequence, choice, all, any, any attribute, open content*.
- Directives: *import, include, redefine, override*.
- Identity constraints: *key, keyref, unique, selector, field, assert*.



Note: The *type alternative, open content, override, and assert* components are available for XML Schema 1.1.

To add a component to the edited schema:

- Click and hold a graphic symbol from the **Palette** view, then drag the component into the **Design** view.
- A line dynamically connects the component with the XML schema structure.
- Release the component into a valid position.



Note: You cannot drop a component into an invalid position. When you hover the component into an invalid position, the mouse cursor changes its shape into . Also, the connector line changes its color from the usual dark grey to the color defined in the **Validation error highlight color** option (default color is red).

To watch our video demonstration about the Schema palette and developing XML Schemas, go to http://oxygentools.com/demo/Schema_Palette.html and http://oxygentools.com/demo/Developing_XML_Schemas.html respectively.

Edit Schema Namespaces

You can use the dialog **XML Schema Namespaces** to easily set a target namespace and define namespace mappings for a newly created XML Schema. In the **Design** mode these namespaces can be modified anytime by choosing **Edit**

Schema Namespaces from the contextual menu. Also you can do that by double-clicking on the schema root in the diagram.

The **XML Schema Namespaces** dialog allows you to edit the following information:

- **Target namespace** - The target namespace of the schema.
- **Prefixes** - The dialog shows a table with namespaces and the mapped prefixes. You can add a new prefix mapping or remove an already existing one.

XML Schema Text Editing Mode

This page is used for editing the XML Schema in a text mode. It offers powerful content completion support, a synchronized Outline view, and multiple *refactoring actions*. The Outline view has two display modes: the *standard outline* mode and the *components* mode.

A diagram of the XML Schema can be presented side by side with the text. To activate the diagram presentation, enable the *Show Full Model XML Schema diagram* check box from the *Diagram* preferences page.

Content Completion

The intelligent **Content Completion Assistant** available in Oxygen XML Editor enables rapid, in-line identification and insertion of elements, attributes and attribute values valid in the editing context. All available proposals are listed in a pop-up list displayed at the current caret position.

The **Content Completion Assistant** is enabled by default. To disable it, *open the Preferences dialog box* and go to **Editor > Content Completion**. It is activated:

- automatically, after a configurable delay from the last key press of the < character. You can adjust the delay *from the Content Completion preferences page*
- on demand, by pressing **Ctrl Space (Command Space on OS X)** on a partial element or attribute name.



Note: If the Content Completion list contains only one valid proposal, when you press the **Ctrl Space (Command Space on OS X)** shortcut key, the proposal is automatically inserted.



Note: You can also start the **Content Completion Assistant** from **Document > Content Completion > Start Content Completion**.

When active, it displays a list of context-sensitive proposals valid at the current caret position. Elements are highlighted in the list using the Up and Down cursor keys on your keyboard. For each selected item in the list, the **Content Completion Assistant** displays a documentation window. You can customize the size of the documentation window by dragging its top, right, and bottom borders.

To insert the selected content:

- press Enter or Tab on your keyboard to insert both the start and end tags.
- press **Ctrl Enter (Command Enter on OS X)** on your keyboard. Oxygen XML Editor inserts both the start and end tags and positions the cursor between the tags, so you can start typing content.

Depending on the *selected schema version*, Oxygen XML Editor populates the proposals list with information taken either from XML Schema 1.0 or 1.1.

Oxygen XML Editor helps you to easily reference a component by providing the list of proposals (complex types, simple types, elements, attributes, groups, attribute groups, or notations) valid in the current context. The components are collected from the current file or from the imported/included schemas.

When editing `xs:annotation/xs:appinfo` elements of an XML Schema, the Content Completion Assistant proposes elements and attributes from a custom schema (by default ISO Schematron). This feature can be configured from the *XSD Content Completion* preferences page.

References to XML Schema Specification

The same as in editing XML documents, the message of an error obtained by validation of an XML Schema document includes a reference to the W3C specification for XML Schema. An error message contains an *Info* field that will open the browser on the "XML Schema Part 1:Structures" specification at exactly the point where the error is described thus allowing you to understand the reason for that error.

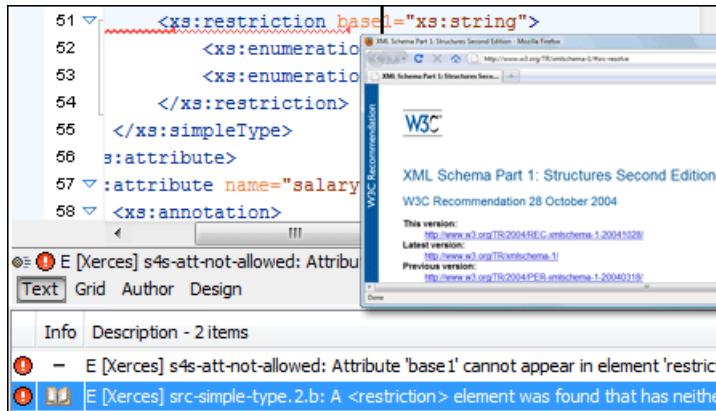


Figure 154: Link to Specification for XML Schema Errors

Validation of an XML Schema containing a type definition with a `minOccurs` or `maxOccurs` attribute having a value larger than 256 limits the value to 256 and issues a warning about this restriction in the Message panel at the bottom of the Oxygen XML Editor window. Otherwise, for large values of the `minOccurs` and `maxOccurs` attributes the validator fails with an *OutOfMemory* error which practically makes Oxygen XML Editor unusable without a restart of the entire application.



Important:

If the schema imports only a namespace without specifying the schema location and a *catalog is set-up* mapping the namespace to a certain location both validation and the schema components outline will correctly identify the imported schema.

XML Schema Actions

- The **Show Definition Ctrl Shift ENTER (Command Shift ENTER on OS X)** action, accessed from the **Document > Schema** menu, moves the cursor to the definition of the referenced XML Schema item. You can also use the **Ctrl Click (Command Click on OS X)** shortcut on a reference to display its definition. The referenced item can be an element, group, simple type or complex type. The same action is executed on a double click on a component name in the *Schema Outline view*. You can define a scope for this action in the same manner you define it for *Search Declarations*.

Highlight Component Occurrences

When a component (for example types, elements, attributes) is found at current cursor position, Oxygen XML Editor performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document. Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is on by default. To configured it, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. A search can also be triggered with the **Search > Search Occurrences in File ()** contextual menu action. All matches are displayed in a separate tab of the **Results** view.

Editing XML Schema in the Master Files Context

Smaller interrelated modules that define a complex XML Schema cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, an element defined in a main schema document is not visible when you edit an included module. Oxygen XML Editor provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You can set a main XML document either using the [master files support from the Project view](#), or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- correct validation of a module in the context of a larger schema structure;
- **Content Completion Assistant** displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one;
- the **Outline** displays the components collected from the entire schema structure;

Searching and Refactoring Actions in XML Schemas

Search Actions

The following search actions can be applied on attribute, attributeGroup, element, group, key, unique, keyref, notation, simple, or complex types and are available from the **Search** submenu in the contextual menu of the current editor or from the **Document > References** menu:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on attribute, attributeGroup, element, group, key, unique, keyref, notation, simple, or complex types and are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

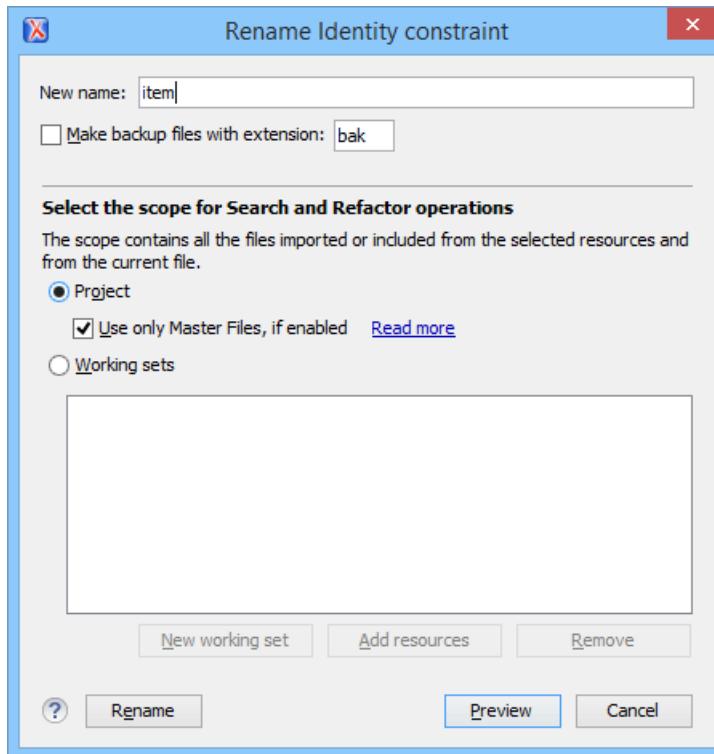


Figure 155: Rename Identity Constraint Dialog Box

Component Dependencies View

The **Component Dependencies** view allows you to spot the dependencies for the selected component of an XML Schema, a [Relax NG schema](#), a [NVDL schema](#) or an [XSLT stylesheet](#). You can open the view from **Window > Show View > Component Dependencies**.

If you want to see the dependencies of a schema component:

- Select the desired schema component in the editor.
- Choose the **Component Dependencies** action from the contextual menu.

The action is available for all named components (for example elements or attributes).

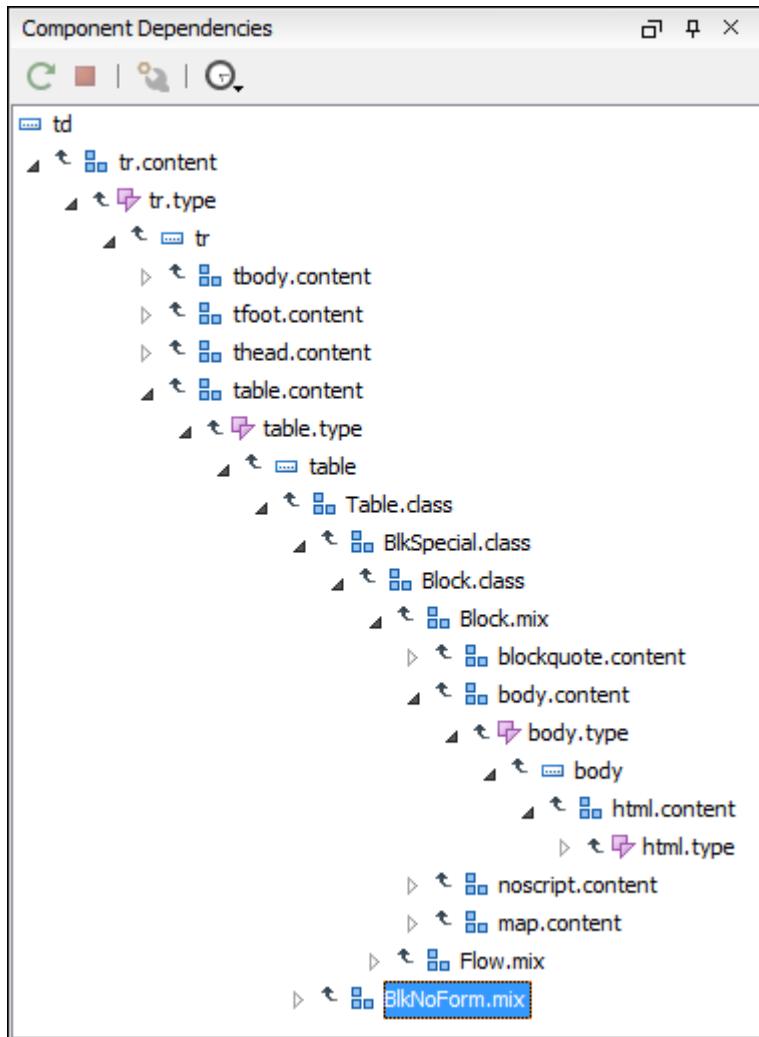


Figure 156: Component Dependencies View - Hierarchy for `xhtml11.xsd`

In the **Component Dependencies** view the following actions are available in the toolbar:

Refresh

Refreshes the dependencies structure.

Stop

Stop the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure.

History

Contains a list of previously executed dependencies computations.

The contextual menu contains the following actions:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.



Tip: If a component contains multiple references to another components, a small table is shown containing all these references. When a recursive reference is encountered, it is marked with a special icon

XML Schema Quick Assist Support

The Quick Assist support improves the development work flow, offering fast access to the most commonly used actions when you edit XML Schema documents.

Quick Assist is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

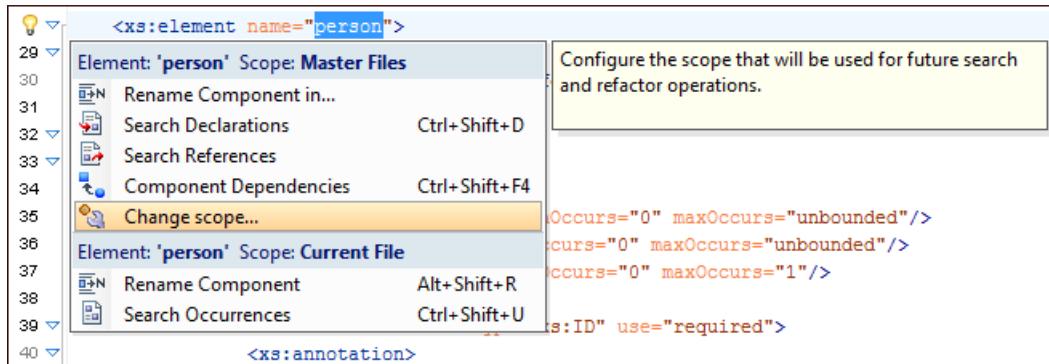


Figure 157: Quick Assist Support

The quick assist support offers direct access to the following actions:

Rename Component in...

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope...

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

To watch our video demonstration about improving schema development using the **Quick Assist** action set, go to http://oxygenvxml.com/demo/Quick_Assist.html.

XML Schema Resource Hierarchy / Dependencies View

The **Resource Hierarchy / Dependencies** view allows you to easily see the hierarchy / dependencies for an XML Schema, a [Relax NG schema](#) or an [XSLT stylesheet](#). To open this view, go to **Window > Show View > Resource Hierarchy / Dependencies**.

The **Resource Hierarchy / Dependencies** is useful when you want to start from an XML Schema (XSD) file and build and review the hierarchy of all the other XSD files that are imported, included or redefined in the given XSD file. The view is also able to build the tree structure, that is the structure of all other XSD files that import, include or redefine the given XSD file. The scope of the search is configurable: the current project, a set of local folders, etc.

The build process for the hierarchy view is started with the **Resource Hierarchy** action available on the contextual menu of the editor panel.

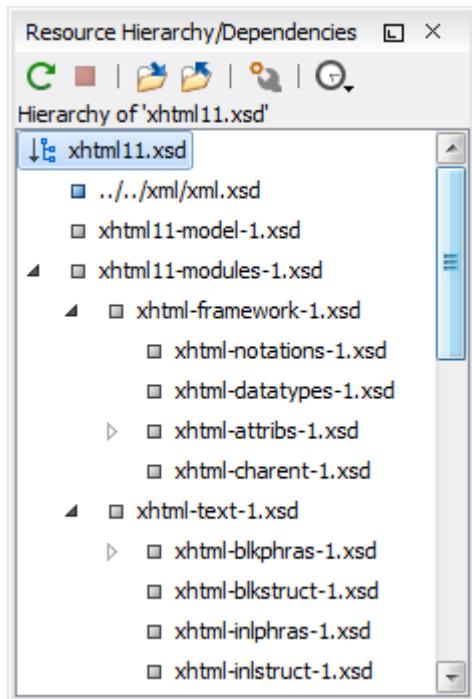


Figure 158: Resource Hierarchy/Dependencies View - Hierarchy for xhtml11.xsd

The build process for the dependencies view is started with the **Resource Dependencies** action available on the contextual menu.

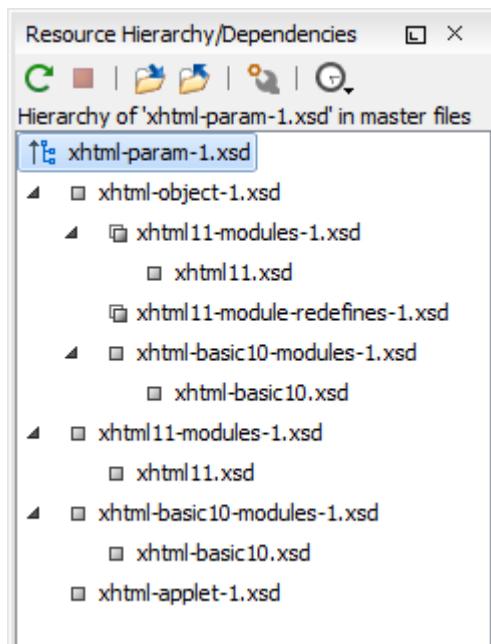


Figure 159: Resource Hierarchy/Dependencies View - Dependencies for xhtml-param-1.xsd

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the  **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

 **Tip:** When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .

 **Note:** The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming XML Schema Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Generating Documentation for an XML Schema

Oxygen XML Editor can generate detailed documentation for the components of an XML Schema in HTML, PDF and DocBook XML formats. You can select the components and the level of detail. The components are hyperlinked in both HTML and DocBook documents.

 **Note:** You can generate documentation both for XML Schema version 1.0 and 1.1.

To generate documentation for an XML Schema document, select **XML Schema Documentation...** from the **Tools > Generate Documentation** menu or from the **Generate Documentation** submenu in the contextual menu of the **Project** view.

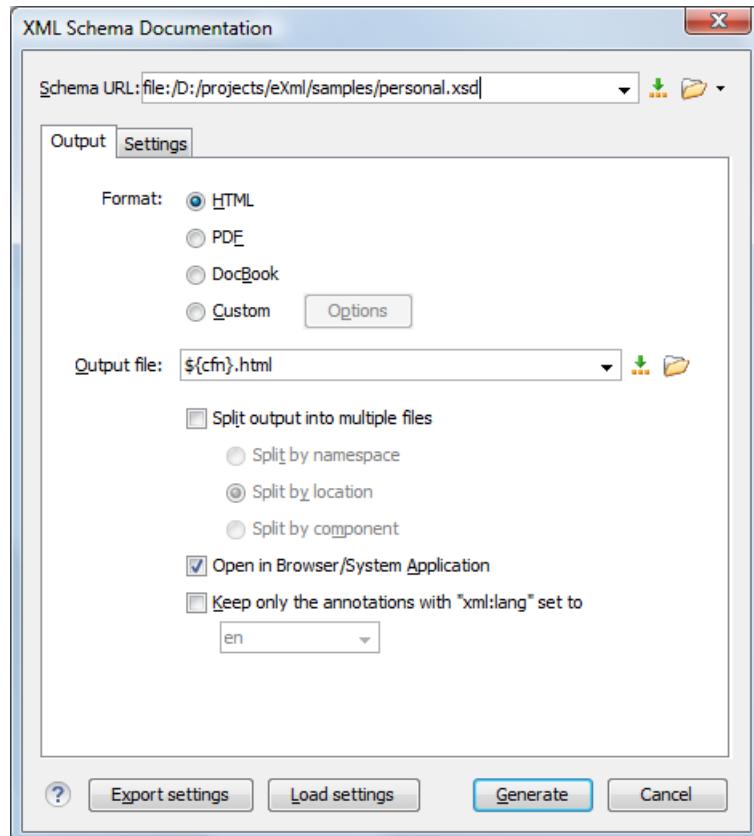


Figure 160: The Output panel of the XML Schema Documentation Dialog Box

The **Schema URL** field of the dialog box must contain the full path to the XML Schema (XSD) file you want to generate documentation for. The schema may be a local or a remote one. You can specify the path to the schema using the editor variables.

The following options are available in the **Settings** tab:

- **Format** - Allows you to choose between three predefined formats (**HTML**, **PDF**, **DocBook**) and a custom one (**Custom**). This allows you to control the output format by providing a custom stylesheet.
- **Output file** - Name of the output file.
- **Split output into multiple files** - Instructs the application to split the output into multiple files. You can choose to split them by namespace, location or component name.
- **Open in Browser/System Application** - Opens the result in the system application associated with the output file type.

 **Note:** To set the browser or system application that will be used, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field. This will take precedence over the default system application settings.

- **Keep only the annotations with xml:lang set to** - The generated output will contain only the annotations with the `xml:lang` attribute set to the selected language. If you choose a primary language code (like `en`, for example), this includes all its possible variations (for example `en-us` and `en-uk` just to name a few).

You can choose to split the output into multiple files by namespace, location or component.

You can export the settings of the **XML Schema Documentation** dialog box to an XML file by pressing the **Export settings** button. With the exported settings file you can generate the same [documentation from the command line interface](#).

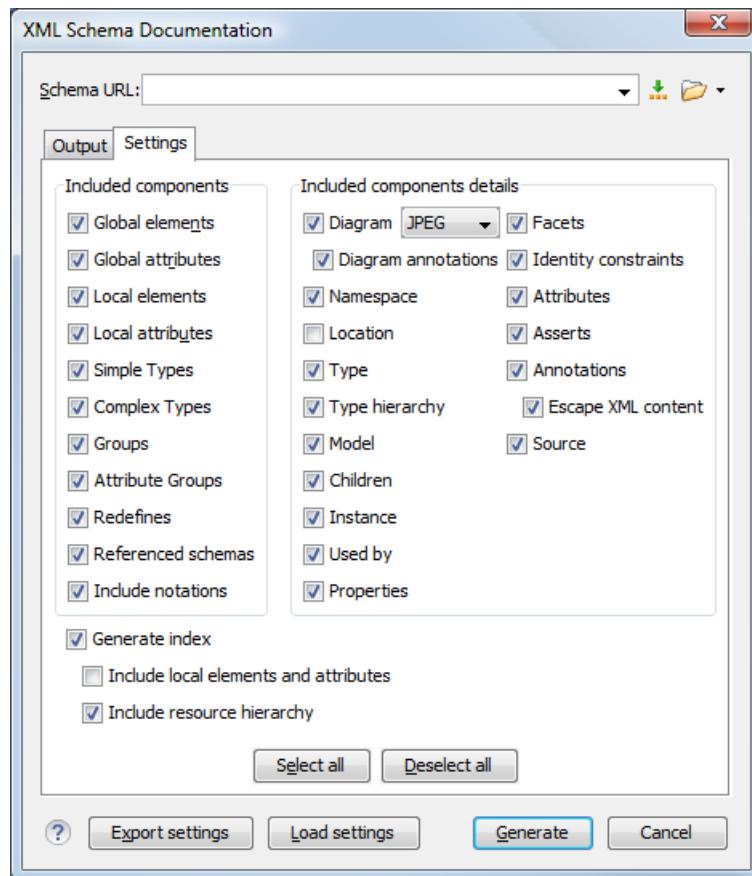


Figure 161: The Settings Panel of the XML Schema Documentation Dialog Box

When you generate documentation for an XML schema you can choose what components to include in the output (global elements, global attributes, local elements, local attributes, simple types, complex types, group, attribute groups, referenced schemas, redefines) and the details to be included in the documentation:

- **Diagram** - Displays the diagram for each component. You can choose the image format (JPEG, PNG, GIF, SVG) to use for the diagram section.

- **Diagram annotations** - This option controls whether the annotations of the components presented in the diagram sections are included.
- **Namespace** - Displays the namespace for each component.
- **Location** - Displays the schema location for each component.
- **Type** - Displays the component type if it is not an anonymous one.
- **Type hierarchy** - Displays the types hierarchy.
- **Model** - Displays the model (sequence, choice, all) presented in BNF form. Different separator characters are used depending on the information item used:
 - **xs : all** - its children will be separated by space characters.
 - **xs : sequence** - its children will be separated by comma characters.
 - **xs : choice** - its children will be separated by / characters.
- **Children** - Displays the list of component's children.
- **Instance** - Displays an XML instance generated based on each schema element.
- **Used by** - Displays the list of all the components that reference the current one. The list is sorted by component type and name.
- **Properties** - Displays some of the component's properties.
- **Facets** - Displays the facets for each simple type
- **Identity constraints** - Displays the identity constraints for each element. For each constraint there are presented the name, type (unique, key, keyref), reference attribute, selector and field(s).
- **Attributes** - Displays the attributes for the component. For each attribute there are presented the name, type, fixed or default value, usage and annotation.
- **Asserts** - Displays the **assert** elements defined in a complex type. The test, XPath default namespace, and annotation are presented for each assert.
- **Annotations** - Displays the annotations for the component. If you choose **Escape XML Content**, the XML tags are present in the annotations.
- **Source** - Displays the text schema source for each component.
- **Generate index** - Displays an index with the components included in the documentation.
- **Include local elements and attributes** - If checked, local elements and attributes are included in the documentation index.
- **Include resource hierarchy** - specifies whether the resource hierarchy for an XML Schema documentation is generated.
- **Export settings / Load settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

These options are persistent between sessions.

Generate Documentation in HTML Format

The HTML documentation contains images corresponding to the schema definitions as the ones displayed by [the schema diagram editor](#). These images are divided in clickable areas which are linked to the definitions of the clicked names of types or elements. The documentation of a definition includes a **Used By** section with links to the other definitions that reference it. If the **Escape XML Content** option is unchecked, the HTML or XHTML tags used inside the **xs : documentation** elements of the input XML Schema for formatting the documentation text (for example ****, **<i>**, **<u>**, ****, ****, etc.) are rendered in the generated HTML documentation.

The generated images format is **PNG**. The image of an XML Schema component contains the graphical representation of that component as it is rendered in [the Schema Diagram panel of the Oxygen XML Editor's XSD editor panel](#).

Table of Contents

Group by: Location

- personal.xsd
 - Elements
 - email
 - family
 - given
 - link
 - name
 - person
 - personnel
 - url
 - Notations
 - gif

XML Schema documentation generated by **<oXygen/>** XML Editor.

Element name

Namespace No namespace

Annotations Name
Annotation

Diagram

Properties Content complex

Used by Element person

Model ALL(family given)
family, given

Children family, given

Instance

```
<name>
  <family>{1,1}</family>
  <given>{1,1}</given>
</name>
```

Source

```
<x:element name="name">
  <x:annotation>
    <x:documentation>Name</x:documentation>
    <x:documentation>Annotation</x:documentation>
  </x:annotation>
  <x:complexType>
    <x:all>
      <x:element ref="family"/>
      <x:element ref="given"/>
    </x:all>
  </x:complexType>
</x:element>
```

Showing:

- Annotations
- Attributes
- Diagrams
- Facets
- Identity Constraints
- Instances
- Model
- Properties
- Source
- Used by

Close

Figure 162: XML Schema Documentation Example

The generated documentation includes a table of contents. You can group the contents by namespace, location, or component type. After the table of contents there is presented some information about the main schema, the imported, included, and redefined schemas. This information contains the schema target namespace, schema properties (attribute form default, element form default, version) and schema location.

Namespace	No namespace
Properties	Attribute Form Default: unqualified Element Form Default: unqualified
Schema location	file:///D:/personal.xsd

Figure 163: Information About a Schema

If you choose to split the output into multiple files, the table of contents is displayed in the left frame. The contents are grouped in the same mode. If you split the output by location, each file contains a schema description and the components that you have chosen to include. If you split the output by namespace, each file contains information about schemas from that namespace and the list with all included components. If you choose to split the output by component, each file contains information about a schema component.

After the documentation is generated you can collapse details for some schema components. This can be done using the **Showing** view:

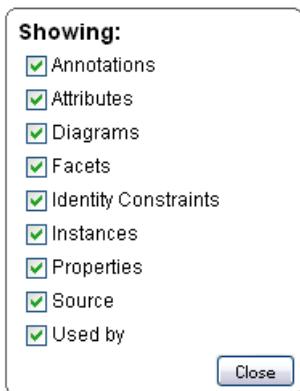


Figure 164: The Showing View

For each component included in the documentation, the section presents the component type follow by the component name. For local elements and attributes, the name of the component is specified as *parent name/component name*. You can easily go to the parent documentation by clicking the parent name.

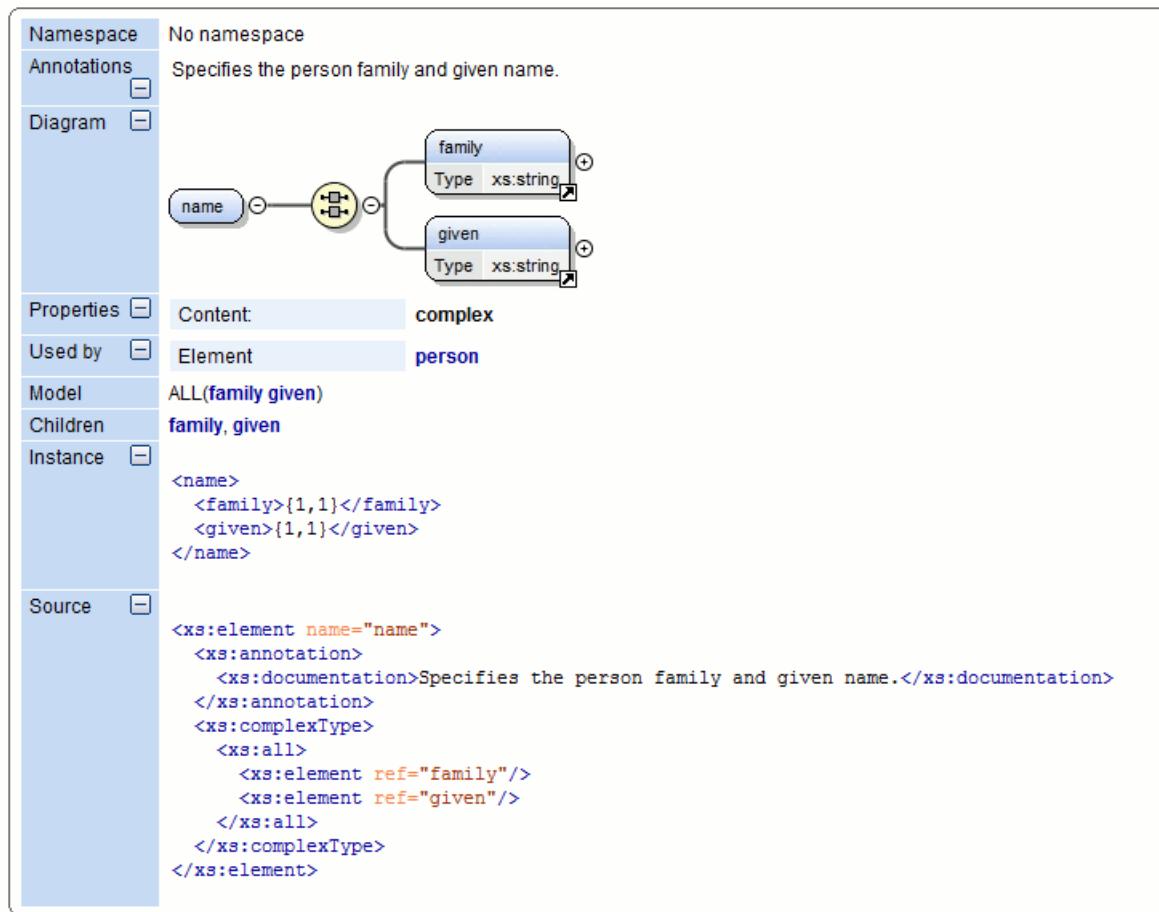
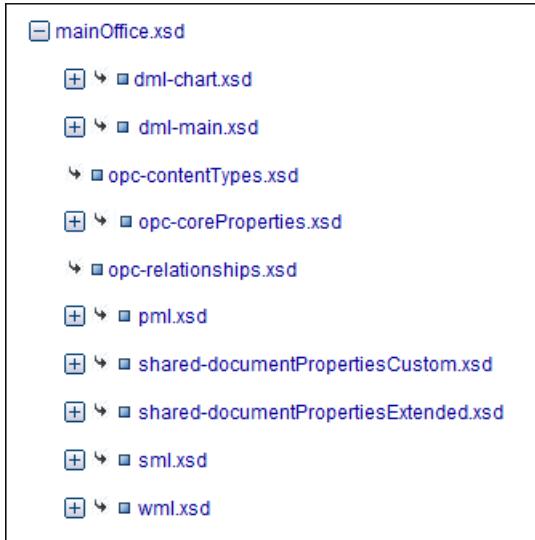


Figure 165: Documentation for a Schema Component

If the schema contains imported or included modules, their dependencies tree is generated in the documentation.



Generate Documentation in PDF, DocBook or a Custom Format

XML Schema documentation can be also generated in PDF, DocBook, or a custom format. You can choose the format from the [Schema Documentation](#) dialog box. For the PDF and DocBook formats, the option to split the output in multiple files is disabled.

When choosing PDF, the documentation is generated in DocBook format and after that a transformation using the FOP processor is applied to obtain the PDF file. To configure the FOP processor, see the [FO Processors](#) preferences page.

If you generate the documentation in DocBook format you can apply a transformation scenario on the output file, for example one of the scenarios proposed by Oxygen XML Editor (*DocBook PDF* or *DocBook HTML*) or configure your own scenario for it.

For the custom format, you can specify your stylesheet to transform the intermediary XML generated in the documentation process. You have to write your stylesheet based on the schema `xsdDocSchema.xsd` from `[OXYGEN_DIR]/frameworks/schema_documentation`. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in `[OXYGEN_DIR]/frameworks/schema_documentation/xsl`.

When using a custom format you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Customizing the Generated PDF

You are able to customize the PDF output of the documentation for an XML schema by running two transformations and customizing the intermediate file. To do this, use the following procedure:

1. Customize the `[OXYGEN_DIR]/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl` stylesheet to include the content that you want to add in the PDF output. Add the content in the XSLT template with the `match="schemaDoc"` attribute between these two elements:

```

<info>
  <pubdate><xsl:value-of select="format-date(current-date(),'[Mn] [D], [Y]', 'en', (), ())"/></pubdate>
</info>

```

```

<xsl:apply-templates select="schemaHierarchy"/>

```



Note: The content that you insert here has to be wrapped in DocBook markup. For details about working with DocBook take a look at the video demonstration from this address
http://www.oxygenxml.com/demo/DocBook_Editing_in_Author.html.

2. Create an intermediary file that holds the content of your XML Schema documentation.
 - a. Go to **Tools > Generate Documentation > XML Schema Documentation**.

- b. Click **Custom** in the **XML Schema Documentation** dialog box.
 - c. Go to **Options**.
 - d. In the **Custom format options** dialog box, enable **Copy additional resources to the output folder**, enter: `[OXYGEN_DIR]/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl` in the **Custom XSL** field and click **OK**.
 - e. When you return to the **Custom format options** dialog box, just press the **Generate** button which will generate a DocBook XML file with an intermediary form of the Schema documentation.
3. Create the final PDF document.
- a. Go to **Document > Transformation > Configure Transformation Scenario** and click **New**.
 - b. In the **New Scenario** dialog box, go to the **XSL URL** field and choose the `[OXYGEN_DIR]/frameworks/docbook/oxygen/xsdDocDocbookCustomizationFO.xsl` file.
 - c. Go to the **FO Processor** tab and enable **Perform FO Processing** and **XSLT result as input**.
 - d. Go to the **Output** tab and select the directory where you want to store the XML Schema documentation output and click **OK**.
 - e. Click **Apply Associated**.

Generating Documentation From the Command-Line Interface

You can export the settings of the **XML Schema Documentation** dialog box to an XML file by pressing the **Export settings** button. With the exported settings file, you can generate the same documentation from the command-line interface by running the following scripts:

- `schemaDocumentation.bat` on Windows.
- `schemaDocumentation.sh` (on OS X / Unix / Linux).

The scripts are located in the Oxygen XML Editor installation folder. The scripts can be integrated in an external batch process launched from the command-line interface.

The script command-line parameter is the relative path to the exported XML settings file. The files which are specified with relative paths in the exported XML settings are made absolute, relative to the folder where the script is ran from.

XML Configuration File

```

<serialized>
  <map>
    <entry>
      <xsdDocumentationOptions>
        <field name="outputFile">
          <String xml:space="preserve">${cfn}.html</String>
        </field>
        <field name="splitMethod">
          <Integer xml:space="preserve">1</Integer>
        </field>
        <field name="openOutputInBrowser">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="format">
          <Integer xml:space="preserve">1</Integer>
        </field>
        <field name="customXSL">
          <null/>
        </field>
        <field name="deleteXMLFiles">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeIndex">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeGlobalElements">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeGlobalAttributes">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
        <field name="includeLocalElements">
          <Boolean xml:space="preserve">true</Boolean>
        </field>
      </xsdDocumentationOptions>
    </entry>
  </map>
</serialized>

```

```

<field name="includeLocalAttributes">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="includeSimpleTypes">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="includeComplexTypes">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="includeGroups">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="includeAttributesGroups">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="includeRedefines">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="includeReferencedSchemas">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsDiagram">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsNamespace">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsLocation">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsType">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsTypeHierarchy">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsModel">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsChildren">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsInstance">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsUsedby">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsProperties">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsFacets">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsAttributes">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsIdentityConstr">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsEscapeAnn">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsSource">
    <Boolean xml:space="preserve">true</Boolean>
</field>
<field name="detailsAnnotations">
    <Boolean xml:space="preserve">true</Boolean>
</field>
</xsdDocumentationOptions>
</entry>
</map>
</serialized>

```

Flatten an XML Schema

You can organize an XML schema on several levels linked by `xs:include` and `xs:import` statements. In some cases, working on such a schema as on a single file is more convenient.

The **Flatten Schema** operation allows you to flatten an entire hierarchy of XML schemas. Starting with the main XML schema, Oxygen XML Editor calculates its hierarchy by processing the `xs:include` and `xs:import` statements. This action is available either from the contextual menu of the editor or from the **Tools** menu.

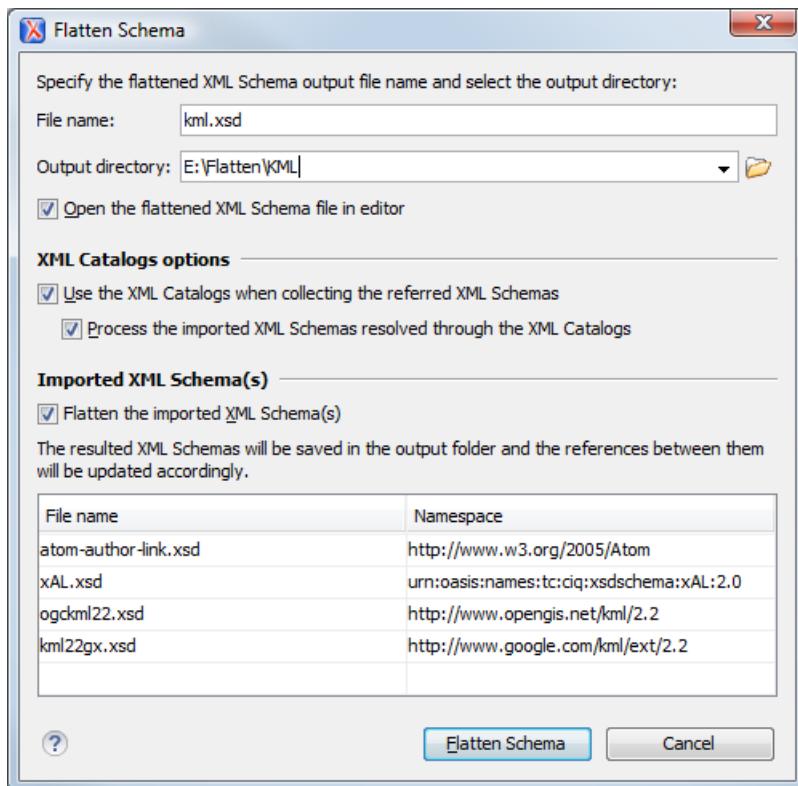


Figure 166: Flatten Schema Dialog

For the main schema file and for each imported schema, a new flattened schema is generated in the output folder. These schemas have the same name as the original ones.

Note: If necessary, the operation renames the resulted schemas to avoid duplicated file names.

Note: You can choose the output folder and the name of each generated schema file.

A flattened XML schema is obtained by recursively adding the components of the included schemas into the main one. This means Oxygen XML Editor replaces the `xs:include`, `xs:redefine`, and `xs:override` elements with the ones coming from the included files.

The following options are available in the **Flatten Schema** dialog:

- **Open the flattened XML Schema file in editor** - opens the main flattened schema in the editing area after the operation completes
- **Use the XML Catalogs when collecting the referenced XML Schemas** - enables resolving the imported and included schemas through the available XML Catalogs;

Note: Changing this option triggers the recalculation of the dependencies graph for the main schema.

- **Process the imported XML Schemas resolved through the XML Catalogs** - specifies whether the imported schemas that were resolved through an XML Catalog are flattened
- **Flatten the imported XML Schema(s)** - specifies whether the imported schemas are flattened.

Note: For the schemas skipped by the flatten operation, no files are created in the output folder and the corresponding import statements remain unchanged.

To flatten a schema from the command line, run one of the following scripts that come with Oxygen XML Editor:

- `flattenSchema.bat` on Windows
- `flattenSchemaMac.sh` on Mac OS X and Unix/Linux

The command line accepts the following parameters:

- `-in:inputSchemaURL` - the input schema URL
- `-outDir:outputDirectory` - the directory where the flattened schemas should be saved
- `-flattenImports:<boolean_value>` - controls if the imported XML Schemas should be flattened or not. The default value `true`
- `-useCatalogs:<boolean_value>` - controls if the references to other XML Schemas should be resolved through the available XML Catalogs. The default value `false`
- `-flattenCatalogResolvedImports:<boolean_value>` - controls if the imported schemas that were resolved through the XML Catalogs should be flattened or not



Note: This option is used only when `-useCatalogs` is set to `true`. The default value is `true`.

- `-verbose` - provides information about the current flatten XML Schema operation
- `--help` | `-help` | `--h` | `-h` - prints the available parameters for the operation

Command Line Example

```
flattenSchema -in:http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd -outDir:mySchemas/ flattened/xhtml
-flattenImports:true -useCatalogs:true -flattenCatalogResolvedImports:true -verbose
```

Generate Sample XML Files

Oxygen XML Editor offers support to generate sample XML files both from XML schema 1.0 and XML schema 1.1, depending on the XML schema version set in **Preferences**.

To generate sample XML files from an XML Schema, use the **Tools > Generate Sample XML Files...** action. This action is also available in the contextual menu of the schema *Design mode*.

The **Generate Sample XML Files** dialog contains the following tabs:

- ***Schema***;
- ***Options***;
- ***Advanced***.

To watch our video demonstration about generating sample XML files, go to
http://oxygentools.com/demo/Generate_Sample_XML_Files.html.

The Schema Tab

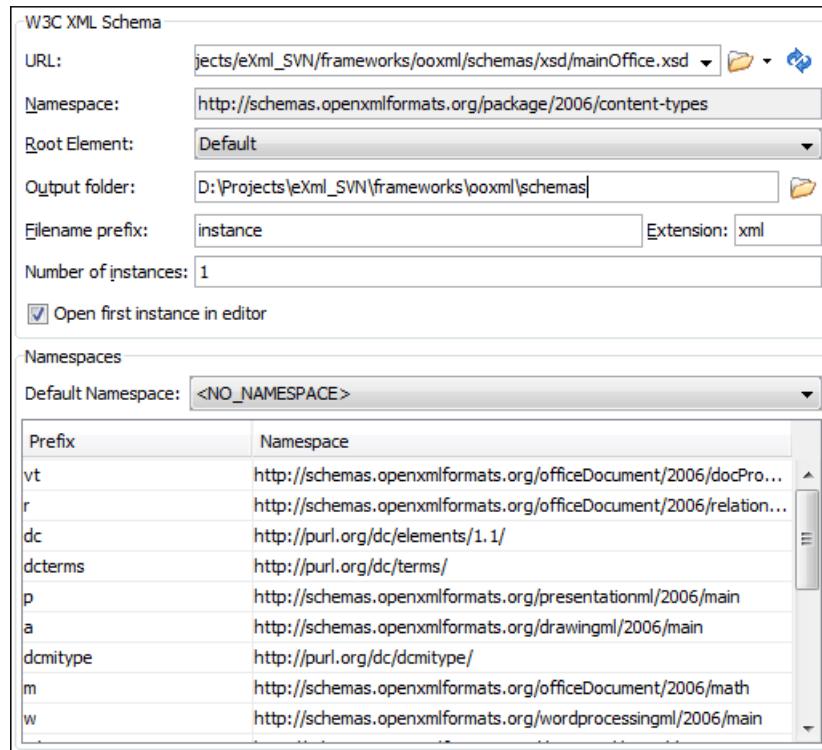


Figure 167: The Generate Sample XML Files Dialog

Complete the dialog as follows:

- **URL** - Schema location as an URL. A history of the last used URLs is available in the drop-down box.
- **Namespace** - Displays the namespace of the selected schema.
- **Document root** - After the schema is selected, this drop-down box is populated with all root candidates gathered from the schema. Choose the root of the output XML documents.
- **Output folder** - Path to the folder where the generated XML instances will be saved.
- **Filename prefix and Extension** - Generated file names have the following format: `prefixN.extension`, where N represents an incremental number from 0 up to *Number of instances* - 1.
- **Number of instances** - The number of XML files to be generated.
- **Open first instance in editor** - When checked, the first generated XML file is opened in editor.
- **Namespaces** - Here you can specify the default namespace as well as the proxies (prefixes) for namespaces.
- **Load settings / Export settings** - The current settings can be saved for further usage with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

The Options Tab

The **Options** tab allows you to set specific options for different namespaces and elements.

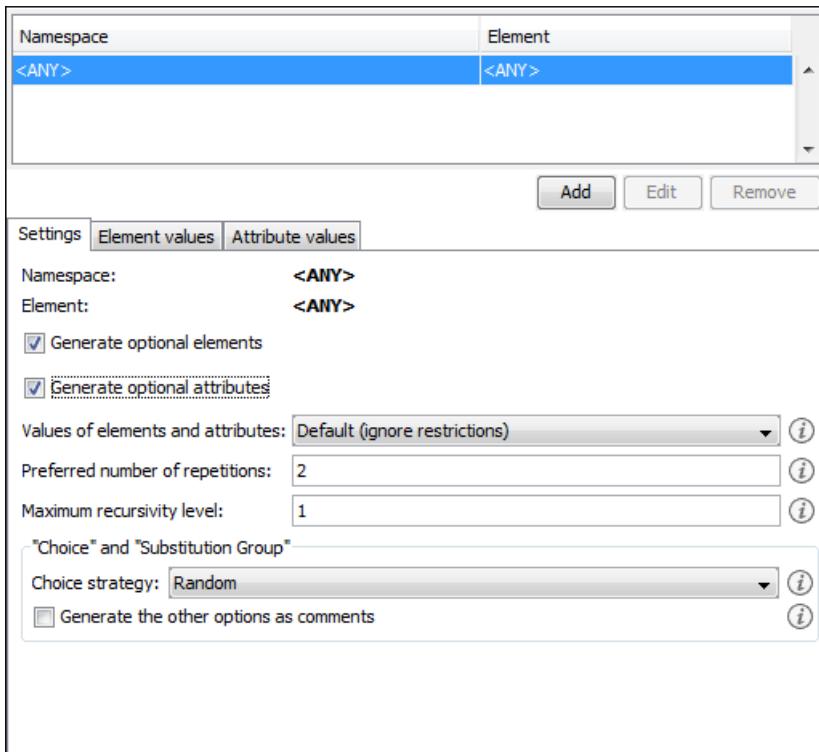


Figure 168: The Generate Sample XML Files Dialog

- **Namespace / Element table** - Allows you to set a namespace for each element name that appears in an XML document instance. The following prefix-to-namespace associations are available:
 - All elements from all namespaces (<ANY> - <ANY>). This is the default setting and can be customized from the [XML Instances Generator](#) preferences page.
 - All elements from a specific namespace.
 - A specific element from a specific namespace.
- **Settings**
 - **Generate optional elements** - When checked, all elements are generated, including the optional ones (having the `minOccurs` attribute set to 0 in the schema).
 - **Generate optional attributes** - When checked, all attributes are generated, including the optional ones (having the `use` attribute set to `optional` in the schema.)
 - **Values of elements and attributes** - Controls the content of generated attribute and element values. Several choices are available:
 - **None** - No content is inserted;
 - **Default** - Inserts a default value depending of data type descriptor of the particular element or attribute. The default value can be either the data type name or an incremental name of the attribute or element (according to the global option from the [XML Instances Generator](#) preferences page). Note that type restrictions are ignored when this option is enabled. For example, if an element is of a type that restricts an `xs:string` with the `xs:maxLength` facet in order to allow strings with a maximum length of 3, the XML instance generator tool may generate string element values longer than 3 characters.
 - **Random** - Inserts a random value depending of data type descriptor of the particular element or attribute.



Important:

If all of the following are true, the [XML Instances Generator](#) outputs invalid values:

- at least one of the restrictions is a regexp;

- the value generated after applying the regexp does not match the restrictions imposed by one of the facets.

This limitation leads to attributes or elements with values set to *Invalid*.

- Preferred number of repetitions** - Allows the user to set the preferred number of repeating elements related with `minOccurs` and `maxOccurs` facets defined in XML Schema.
 - If the value set here is between `minOccurs` and `maxOccurs`, then that value is used;
 - If the value set here is less than `minOccurs`, then the `minOccurs` value is used;
 - If the value set here is greater than `maxOccurs`, then `maxOccurs` is used.
- Maximum recursion level** - If a recursion is found, this option controls the maximum allowed depth of the same element.
- Choice strategy** - Option used in case of `xs:choice` or `substitutionGroup` elements. The possible strategies are:
 - First** - the first branch of `xs:choice` or the head element of `substitutionGroup` is always used;
 - Random** - a random branch of `xs:choice` or a substitute element or the head element of a `substitutionGroup` is used.
- Generate the other options as comments** - Option to generate the other possible choices or substitutions (for `xs:choice` and `substitutionGroup`). These alternatives are generated inside comments groups so you can uncomment and use them later. Use this option with care (for example on a restricted namespace and element) as it may generate large result files.
- Load settings / Export settings** - The current settings can be saved for further usage with the **Export settings** button, and reloaded when necessary with the **Load settings** button.
- Element values** - The **Element values** tab allows you to add values that are used to generate the elements content. If there are more than one value, then the values are used in a random order.
- Attribute values** - The **Attribute values** tab allows you to add values that are used to generate the attributes content. If there are more than one value, then the values are used in a random order.

The Advanced Tab

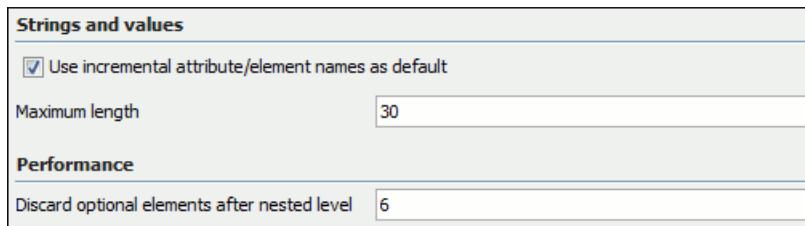


Figure 169: Advanced Tab

This tab allows you to set advanced options that controls the output values of elements and attributes.

- Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an `a` element the generated values are: `a1, a2, a3`, etc. If not checked, the value is the name of the type of that element / attribute, for example: `string, decimal`, etc.
- Maximum length** - The maximum length of string values generated for elements and attributes.
- Discard optional elements after nested level** - The optional elements that exceed the specified nested level are discarded. This option is useful for limiting deeply nested element definitions that can quickly result in very large XML documents.

Running the Generate Sample XML Files Action from Command Line

The **Generate Sample XML Files** tool can be also used from command line by running the script called `xmlGenerator.bat` (on Windows) / `xmlGenerator.sh` (on Mac OS X / Unix / Linux) located in the Oxygen XML Editor installation folder. The parameters can be set once in the dialog, exported to an XML file on disk with the

button **Export settings** and reused from command line. With the exported settings file you can generate the same XML instances from the command line as from the dialog:

```
xmlGenerator.bat path_of_CFG_file
```

The script can be integrated in an external batch process launched from the command line. The command line parameter of the script is the relative path to the exported XML settings file. The files specified with relative paths in the exported XML settings will be made absolute relative to the folder where the script is run.

The following example shows such an XML configuration file:

XML Configuration File

```
<settings>
  <schemaSystemId>http://www.w3.org/2001/XMLSchema.xsd</schemaSystemId>
  <documentRoot>schema</documentRoot>
  <outputFolder>D:\projects\output</outputFolder>
  <filenamePrefix>instance</filenamePrefix>
  <filenameExtension>xml</filenameExtension>
  <noOfInstances>1</noOfInstances>
  <openFirstInstance>true</openFirstInstance>
  <defaultNamespace>&lt;NO_NAMESPACE&gt;</defaultNamespace>
  <element namespace="&lt;ANY;> name="&lt;ANY;>"
    <generateOptionalElements>false</generateOptionalElements>
    <generateOptionalAttributes>false</generateOptionalAttributes>
    <valuesForContentType>DEFAULT</valuesForContentType>
    <preferredNumberOfRepetitions>2</preferredNumberOfRepetitions>
    <maximumRecursivityLevel>1</maximumRecursivityLevel>
    <choicesAndSubstitutions strategy="RANDOM"
      generateOthersAsComments="false"/>
    <attribute namespace="&lt;ANY;> name="&lt;ANY;>"
      <name>&lt;ANY;></name>
      <attributeValue>attrValue1</attributeValue>
      <attributeValue>attrValue2</attributeValue>
    </attribute>
  </element>
  <element namespace="&lt;NO_NAMESPACE;>
    name="&lt;ANY;>"
    <generateOptionalElements>true</generateOptionalElements>
    <generateOptionalAttributes>true</generateOptionalAttributes>
    <valuesForContentType>DEFAULT</valuesForContentType>
    <preferredNumberOfRepetitions>2</preferredNumberOfRepetitions>
    <maximumRecursivityLevel>1</maximumRecursivityLevel>
    <choicesAndSubstitutions strategy="RANDOM"
      generateOthersAsComments="true"/>
    <elementValue>value1</elementValue>
    <elementValue>value2</elementValue>
    <attribute namespace="&lt;ANY;> name="&lt;ANY;>"
      <name>&lt;ANY;></name>
      <attributeValue>attrValue1</attributeValue>
      <attributeValue>attrValue2</attributeValue>
    </attribute>
  </element>
</settings>
```

XML Schema Regular Expressions Builder

The XML Schema regular expressions builder allows testing regular expressions on a fragment of text as they are applied to an XML instance document. Start the tool from the **Tools** menu.

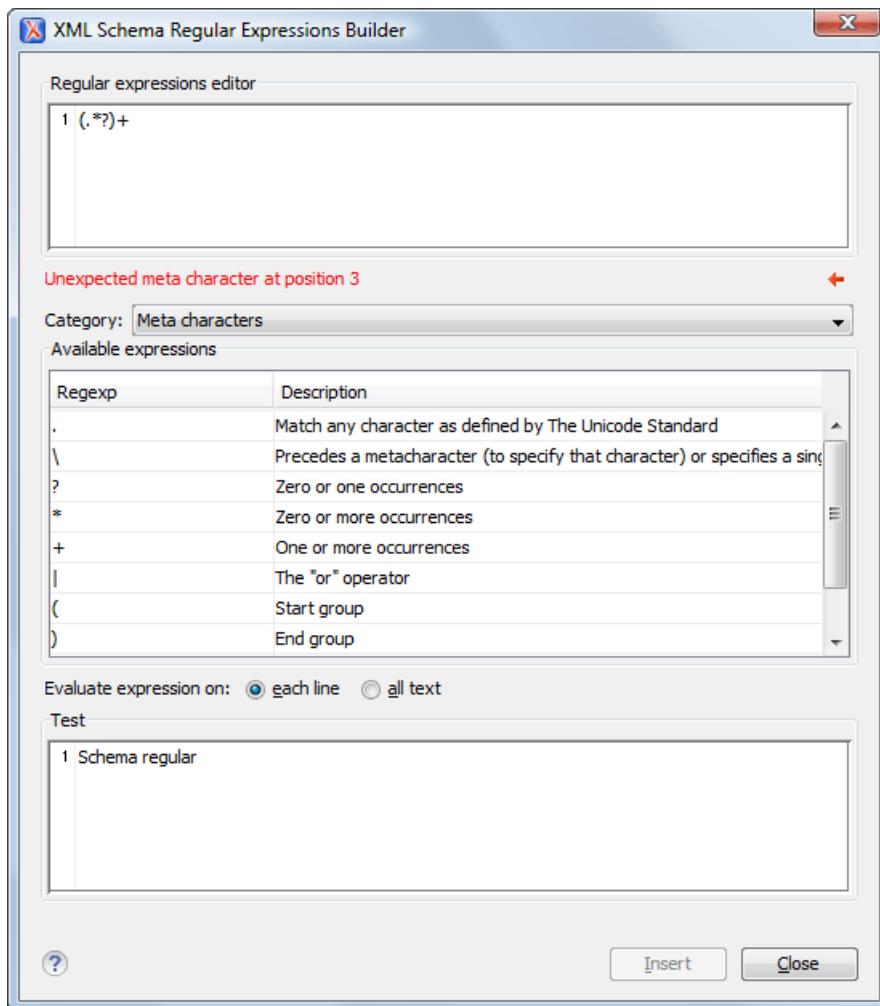


Figure 170: XML Schema Regular Expressions Builder Dialog Box

The dialog box contains the following sections:

- **Regular expressions editor** - allows you to edit the regular expression to be tested and used. Content completion is available and presents a list with all the predefined expressions. It is triggered by pressing **Ctrl Space (Command Space on OS X)**.
- **Error display area** - if the edited regular expression is incorrect, an error message will be displayed here. The message contains the description and the exact location of the error. Also, a click on the quick navigation button () highlights the error inside the regular expression.
- **Category** combo box - here you can choose from several categories of predefined expressions. The selected category influences the displayed expressions in the **Available expressions** table.
- **Available expressions** table - holds the available regular expressions and a short description for each of them. The set of expressions depends on the category selected in the previous combo box. You can add an expression in the **Regular expressions editor** by double-clicking on the expression row in the table. You will notice that in the case of **Character categories** and **Block names** the expressions are also listed in complementary format. For example: $\text{\p{Lu}}$ - Uppercase letters; $\text{\P{Lu}}$ - Complement of: Uppercase letters.
- **Evaluate expression on** radio buttons - there are available two options:
 - **Evaluate expression on each line** - the edited expression will be applied on each line in the **Test** area.
 - **Evaluate expression on all text** - the edited expression will be applied on the whole text.
- **Test** area - a text editor which allows you to enter a text sample on which the regular expression will be applied. All matches of the edited regular expression will be highlighted.

After editing and testing your regular expression you can insert it in the current editor. The **Insert** button will become active when an editor is opened in the background and there is an expression in the **Regular expressions editor**.

The regular expression builder cannot be used to insert regular expressions in *the grid version* or *the schema version* of a document editor. Accordingly, the **Insert** button of the dialog box will be disabled if the current document is edited in grid mode.



Note: Some regular expressions may block indefinitely the Java Regular Expressions engine. If the execution of the regular expression does not end in about five seconds, the application displays a dialog box that allows you to interrupt the operation.

Create an XML Schema From a Relational Database Table

To create an XML Schema from the structure of a relational database table use *the special wizard available in the Tools menu*.

XML Schema 1.1

Oxygen XML Editor offers full support for XML Schema 1.1, including:

- XML Documents Validation and Content Completion Based on XML Schema 1.1;
- XML Schema 1.1 Validation and Content Completion;
- Editing XML Schema 1.1 files in the Schema Design mode;
- The Flatten Schema action;
- Resource Hierarchy/Dependencies and Refactoring Actions;
- Master Files;
- Generating Documentation for XML Schema 1.1;
- Support for generating XML instances based on XML Schema.

XML Schema 1.1 is a superset of XML Schema 1.0, that offers lots of new powerful capabilities.

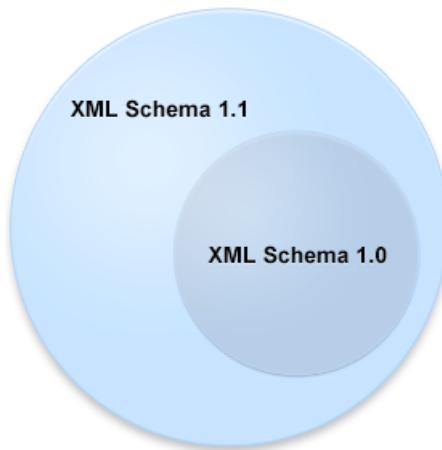


Figure 171: XML Schema 1.1

The significant new features in XSD 1.1 are:

- **Assertions** - support to define assertions against the document content using XPath 2.0 expressions (an idea borrowed from Schematron);
- **Conditional type assignment** - the ability to select the type against which an element is validated based on the values of the attribute of the element;
- **Open content** - content models are able to use the `openContent` element to specify content models with *open content*. These content models allow elements not explicitly mentioned in the content model to appear in the document instance. It is as if wildcards were automatically inserted at appropriate points within the content model. A schema document wide default may be set, which causes all content models to be open unless specified otherwise.

To see the complete list with changes since version 1.0, go to http://www.w3.org/TR/xmlschema11-1/#ch_specs.

XML Schema 1.1 is intended to be mostly compatible with XML Schema 1.0 and to have approximately the same scope. It also addresses bug fixes and brings improvements that are consistent with the constraints on scope and compatibility.

-  **Note:** An XML document conforming to a 1.0 schema can be validated using a 1.1 validator, but an XML document conforming to a 1.1 schema may not validate using a 1.0 validator.

In case you are constrained to use XML Schema 1.0 (for example if you develop schemas for a server that uses an XML Schema 1.0 validator which cannot be updated), change the default XML Schema version to 1.0. If you keep the default XML Schema version set to 1.1, the content completion window presents XML Schema 1.1 elements that you can insert accidentally in an 1.0 XML Schema. So even if you make a document invalid conforming with XML Schema 1.0, the validation process does not signal any issues.

To change the default XML Schema version, [open the Preferences dialog box](#) and go to **XML > XML Parser > XML Schema**.

To watch our video demonstration about the XML Schema 1.1 support, go to http://oxygengxml.com/demo/XML_Schema_11.html.

Setting the XML Schema Version

Oxygen XML Editor lets you set the version of the XML Schema you are editing either in the **XML Schema** preferences page, or through the versioning attributes. In case you want to use the versioning attributes, set the `minVersion` and `maxVersion` attributes, from the <http://www.w3.org/2007/XMLSchema-versioning> namespace, on the `schema` root element.

-  **Note:** The versioning attributes take priority over the XML Schema version defined in the preferences page.

Table 8: Using the `minVersion` and `maxVersion` Attributes to Set the XML Schema Version

Versioning Attributes	XML Schema Version
<code>minVersion = "1.0" maxVerion = "1.1"</code>	1.0
<code>minVersion = "1.1"</code>	1.1
<code>minVersion = "1.0" maxVerion = greater than "1.1"</code>	the XML Schema version defined in the XML Schema preferences page.
Not set in the XML Schema document.	the XML Schema version defined in the XML Schema preferences page.

To change the XML Schema version of the current document, use the **Change XML Schema version** action from the contextual menu. This is available both in the **Text** mode, and in the **Design** mode and opens the **Change XML Schema version** dialog box. The following options are available:

- **XML Schema 1.0** - Inserts the `minVersion` and `maxVersion` attributes on the `schema` element and gives them the values "1.0" and "1.1" respectively. Also, the namespace declaration (`xmlns:vc=http://www.w3.org/2007/XMLSchema-versioning`) is inserted automatically in case it does not exist.
- **XML Schema 1.1** - Inserts the `minVersion` attribute on the `schema` element and gives it the value "1.1". Also, removes the `maxVersion` attribute if it exists and adds the versioning namespace (`xmlns:vc=http://www.w3.org/2007/XMLSchema-versioning`) in case it is not declared.
- **Default XML Schema version** - Removes the `minVersion` and `maxVersion` attributes from the `schema` element. The default schema version, defined in the **XML Schema** preferences page, is used.

-  **Note:** The **Change XML Schema version** action is also available in the informative panel presented at the top of the edited XML Schema. In case you close this panel, it will no longer appear until you restore Oxygen XML Editor to its default options.

Oxygen XML Editor automatically uses the version set through the `versioning` attributes, or the default version in case the `versioning` attributes are not declared, when proposing content completion elements and validating an XML Schema. Also, the XML instance validation against an XML Schema is aware of the `versioning` attributes defined in the XML Schema.

When you generate sample XML files from an XML Schema, Oxygen XML Editor takes into account the `minVersion` and `maxVersion` attributes defined in the XML Schema.

Linking Between Development and Authoring

The **Author** mode is available on the XML Schema editor allowing you to edit visually the schema annotations. It presents a polished and compact view of the XML Schema, with support for links on imported/included schemas. Embedded Schematron is supported only in Relax NG schemas with XML syntax.

Editing XQuery Documents

This section explains the features of the XQuery editor and how to use them.

XQuery Outline View

The XQuery document structure is presented in the **XQuery Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to a component by knowing its name. It can be opened from the **Window > Show View > Outline** menu action.

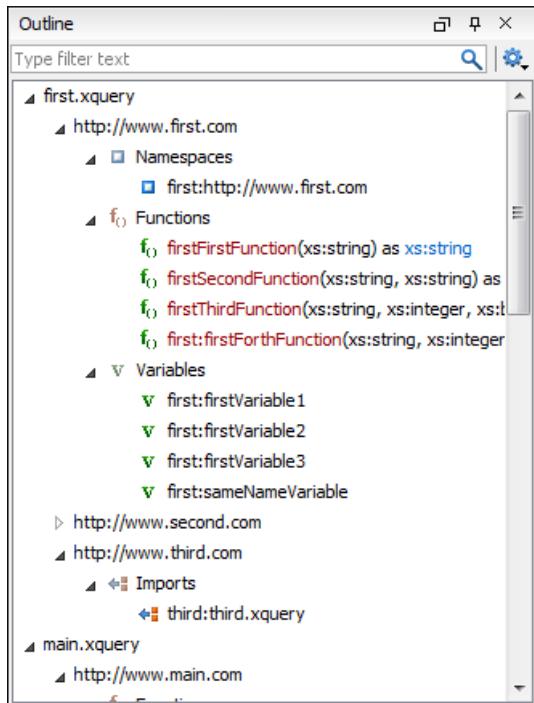


Figure 172: XQuery Outline View

The following actions are available in the **Settings** menu on the Outline view's toolbar:

◀ Selection update on caret move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret moves or the changes performed in the XQuery editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Sort

Allows you to alphabetically sort the XQuery components.

Show all components

Displays all collected components starting from the current file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the **Outline** view.

If you know the component name, you can search it in the **Outline** view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, (**Enter**), (**Tab**), (**Shift-Tab**). To switch from tree structure to the filter text field, you can use (**Tab**), (**Shift-Tab**).

 **Tip:** The search filter is case insensitive. The following wildcards are accepted:

- * - any string
- ? - any character
- , - patterns separator

If no wildcards are specified, the string to search is used as a partial match (like `*textToFind*`).

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Folding in XQuery Documents

In a large XQuery document, the instructions enclosed in the '{' and '}' characters can be collapsed so that only the needed instructions remain in focus. The same *folding features available for XML documents* are also available in XQuery documents.

```

8 let $minRating := min($review/reviews/review[@movie-id = $movie-id]/rating) ↵
9 return ↵
10 <movie id="${movie/@id}"> ↵
11   {$movie/title} ↵
12   {$movie/year} ↵
13   <avgRating> ↵
14   { ↵
15     if ($avgRating) then $avgRating else "not rated" ↵
16   } ↵
17 </avgRating> ↵
18   <maxRating> ↵
19     <value> ↵
20   ▶   { ↵ [2 lines] ↵
21     </value> ↵
22   ▶   { ↵ [5 lines] ↵
23     </maxRating> ↵
24   ▶   <minRating> ↵
25     <value> ↵
26   ▶   { ↵ [2 lines] ↵
27     </value> ↵
28   ▶   { ↵ [5 lines] ↵
29     </minRating> ↵
30   </movie> ↵

```

Figure 173: Folding in XQuery Documents

There is available the action **Go to Matching Bracket Ctrl Shift G (Command Shift G on OS X)** on contextual menu of XQuery editor for going to matching character when cursor is located at '{' character or '}' character. It helps for finding quickly matching character of current folding element.

Formatting and Indenting XQuery Documents

Editing XQuery documents may lead to large chunks of content that is not easily readable by human audience. Also, each developer may have a particular way of writing XQuery code. Oxygen XML Editor assists you in maintaining a consistent code writing style by providing the **Format and Indent** action.

The **Format and Indent** action achieves this by performing the following steps:

- manages whitespaces, by collapsing or inserting space characters where needed.
- formats complex expressions on multiple, more readable lines by properly indenting each of them. The amount of whitespaces that form an indent unit is controlled through one of the **Indent with tabs** and **Indent size** options from the [Format Preferences page](#).



Note: These operations can be performed only if your XQuery document conforms with W3C XQuery 1.0, XQuery Update Facility 1.0, and XQuery 3.0 specifications. If the **Format and Indent** operation fails, the document is left unaltered and an error message is presented in the **Results** view.

The **Format and Indent** action is available in the **Document > Source** menu and also on the toolbar.

Generating HTML Documentation for an XQuery Document

To generate HTML documentation for an XQuery document, use the **XQuery Documentation** dialog box. It is opened with the **XQuery Documentation...** action that is available from the **Tools > Generate Documentation** menu or from the **Generate Documentation** submenu in the contextual menu of the **Project** view.

The dialog box allows you to configure a set of parameters of the process of generating the HTML documentation.

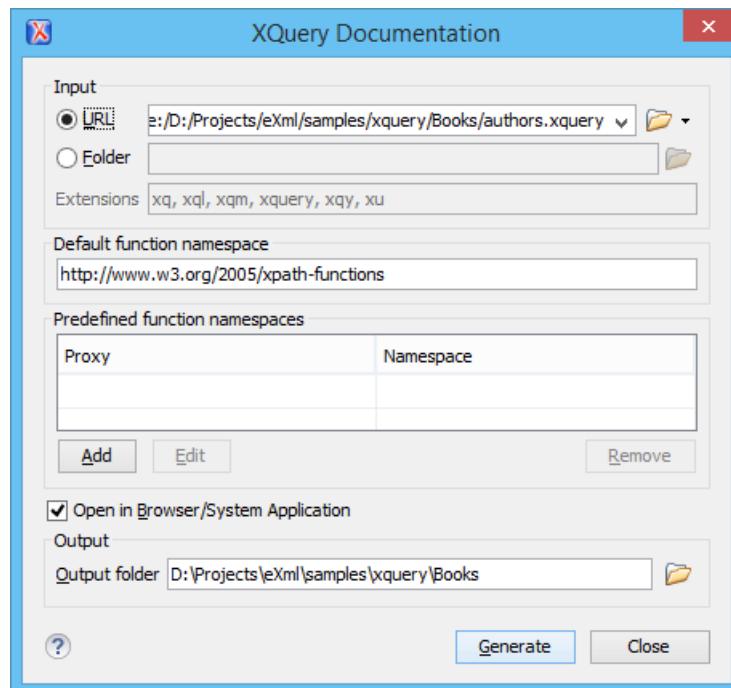


Figure 174: The XQuery Documentation Dialog Box

The following options are available:

- Input** - The **Input** panel allows the user to specify either the **File** or the **Folder** which contains the files for which to generate the documentation. One of the two text fields of the **Input** panel must contain the full path to the XQuery

file. Extensions for the XQuery files contained in the specified directory can be added as comma-separated values. Default there are offered `xquery`, `xq`, `xqy`.

- **Default function namespace** - Optional URI for the default namespace for the submitted XQuery, only if it exists.
- **Predefined function namespaces** - Optional engine dependent, predefined namespaces that the submitted XQuery refers to. They allow the conversion to generate annotation information to support the presentation component hypertext linking, only if the predefined modules have been loaded into the local xqDoc XML repository.
- **Open in Browser/System Application** - Select this option if you want the result to be opened in the system application associated with that file type.

 **Note:** To set the browser or system application that will be used, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field. This will take precedence over the default system application settings.

- **Output** - Allows the user to specify where the generated documentation is saved on disk.

Editing WSDL Documents

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

Oxygen XML Editor provides a special type of editor dedicated to WSDL documents. The WSDL editor offers support to check whether a WSDL document is valid, a specialized Content Completion Assistant, a component oriented **Outline** view, searching and refactoring operations, and support to generate documentation.

Both WSDL version 1.1 and 2.0 are supported and SOAP versions 1.1 and 1.2. That means that in the location where a SOAP extension can be inserted the **Content Completion Assistant** offers elements from both SOAP 1.1 and SOAP 1.2. Validation of SOAP requests is executed first against a SOAP 1.1 schema and then against a SOAP 1.2 schema. In addition to validation against the XSD schemas, Oxygen XML Editor also checks if the WSDL file conforms with the WSDL specification (available only for WSDL 1.1 and SOAP 1.1).

In the following example you can see how the errors are reported.

```

97   <output name="getVersion3Out">
98     <soap:body use="encoded" namespace="http://arcweb.esri.com/services/N2/GetVersion3">
99   </output>
100  <soap:address location="http://arcweb.esri.com/services/N2/GetVersion3" />
101 </operation>
102 </binding>

```

Description - 1 item Resource
E cvc-complex-type.2.4.a: Invalid content was found ... PlaceFinderSample

[E cvc-complex-type.2.4.a: Invalid content was found starting with element 'soap:address'. One of {<http://schemas.xmlsoap.org/wsdl/>:fault}' is expected.]

Figure 175: Validating a WSDL file

To watch our video demonstration about the WSDL editing support in Oxygen XML Editor, go to http://www.oxygenxml.com/demo/Create_New_WSDL.html.

WSDL Outline View

The WSDL **Outline** view displays the list of all the components (services, bindings, port types and so on) of the currently open WSDL document along with the components of its imports.

In case you use the [**Master Files support**](#), the **Outline** view collects the components of a WSDL document starting from the master files of the current document.

To enable the **Outline** view, go to **Window > Show View > Outline**.

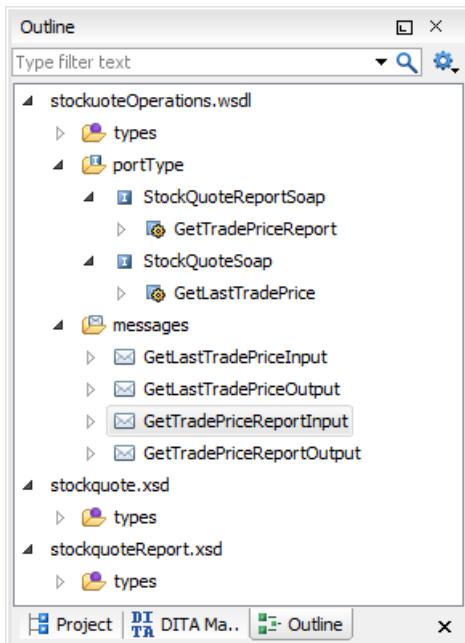


Figure 176: The WSDL Outline View

The **Outline** view can display both the components of the current document and its XML structure, organized in a tree like fashion. You can switch between the components display mode and the XML structure display mode using the

Show XML structure and **Show components** actions. The following actions are available in the **Settings** menu on the Outline view toolbar when you work with the components display mode:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches.

Selection update on caret move

Controls the synchronization between the **Outline** view and the current document. The selection in the **Outline** view can be synchronized with the caret moves or the changes in the WSDL editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the current document.

Show XML structure

Displays the XML structure of the current document in a tree-like structure.

Sort

Sorts the components in the **Outline** view alphabetically.

Show all components

Displays all the components that were collected starting from current document or from the main document in case it is defined.

Show referable components

Displays all the components that you can reference from the current document.

Show only local components

Displays the components defined in the current file only.

Group by location

Groups the WSDL components by their location.

Group by type

Groups the WSDL components by their type.

Group by namespace

Groups the WSDL components by their namespace.



Note: By default all the three grouping criteria are active.

When you work with the XML structure display mode the following actions are available:

Show components

Switches the **Outline** view to the components display mode.

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).

Configure displayed attributes

Displays the [XML Structured Outline preferences page](#).

The following contextual menu actions are available in the **Outline** view when you use it in the components display mode:

Edit Attributes

Opens a dialog that allows you to edit the attributes of the currently selected component.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Search references

Searches for the references of the currently selected component.

Search references in

Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies

Displays the dependencies of the currently selected component.

Resource Hierarchy

Displays the hierarchy for the currently selected resource.

Resource Dependencies

Displays the dependencies of the currently selected resource.

Rename Component in...

Renames the currently selected component in the context of a scope that you define.

The following contextual menu actions are available in the **Outline** view when you use it in the XML structure display mode:

Append Child

Displays a list of elements that you can insert as children of the current element.

Insert Before

Displays a list of elements that you can insert as siblings of the current element, before the current element.

Insert After

Displays a list of elements that you can insert as siblings of the current element, after the current element.

<!-- Toggle Comment

Comments/uncomments the currently selected element.

Search references

Searches for the references of the currently selected component.

Search references in

Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies

Displays the dependencies of the currently selected component.

Rename Component in...

Renames the currently selected component in the context of a scope that you define.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand more

Expands the structure of a component in the **Outline** view.

Collapse all

Collapses the structure of all the component in the **Outline** view.

To switch from the tree structure to the text filter, use **Tab** and **Shift-Tab**.



Tip: The search filter is case insensitive. The following wildcards are accepted:

- * - any string
- ? - any character
- , - patterns separator

If no wildcards are specified, the string to search is used as a partial match (like `*textToFind*`).

The **Outline** content and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

Content Completion in WSDL Documents

The **Content Completion Assistant** is a powerful feature that enhances the editing of WSDL documents. It helps you define WSDL components by proposing context-sensitive element names. Another important capability of the **Content Completion Assistant** is to propose references to the defined components when you edit attribute values. For example, when you edit the `type` attribute of a binding element, the **Content Completion Assistant** proposes all the defined port types. Each proposal that the **Content Completion Assistant** offers is accompanied by a documentation hint.



Note: XML schema specific elements and attributes are offered when the current editing context is the internal XML schema of a WSDL document.

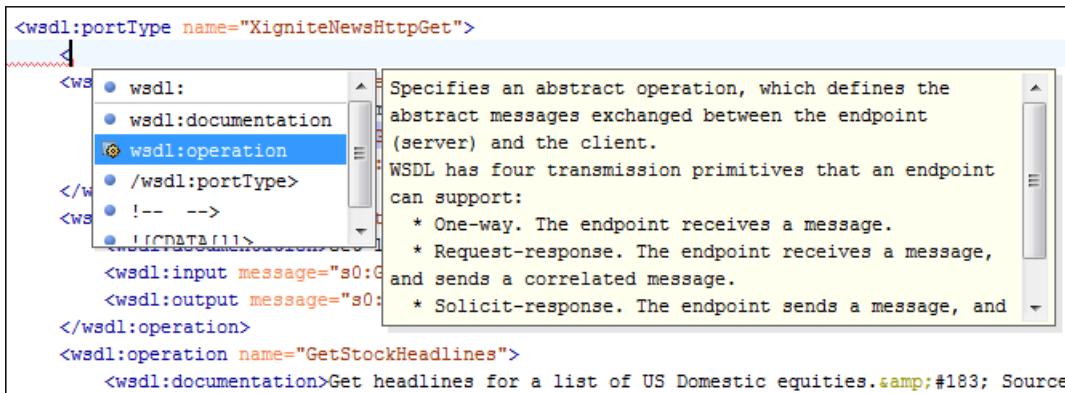


Figure 177: WSDL Content Completion Window

Note: The **Content Completion Assistant** collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to [Defining Master Files at Project Level](#).

Namespace prefixes in the scope of the current context are presented at the top of the content completion window to speed up the insertion into the document of prefixed elements.



Figure 178: Namespace Prefixes in the Content Completion Window

For the common namespaces, like XML Schema namespace (<http://www.w3.org/2001/XMLSchema>) or SOAP namespace (<http://schemas.xmlsoap.org/wsdl/soap/>), Oxygen XML Editor provides an easy mode to declare them by proposing a prefix for these namespaces.

Editing WSDL Documents in the Master Files Context

Smaller interrelated modules that define a complex WSDL structure cannot be correctly edited or validated individually, due to their interdependency with other modules. Oxygen XML Editor provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger WSDL structure.

You can set a main WSDL document either using the [master files support from the Project view](#), or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main WSDL document. In this case, it considers the current module as the main WSDL document.

The advantages of editing in the context of a master file include:

- correct validation of a module in the context of a larger WSDL structure;
- **Content Completion Assistant** displays all components valid in the current context;
- the **Outline** displays the components collected from the entire WSDL structure.

Note: When you edit an XML schema document that has a WSDL document set as master, the validation operation is performed over the master WSDL document.

To watch our video demonstration about editing WSDL documents in the master files context, go to http://oxygenvxml.com/demo/WSDL_Working_Modules.html.

Searching and Refactoring Operations in WSDL Documents

Search Actions

The following search actions are available from the **Search** submenu in the contextual menu of the current editor or from the **Document > References** menu:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

The following action is available from the **Document > Schema** menu:

-  **Show Definition** - Takes you to the location of the definition of the current item.
-  **Note:** You can also use the **Ctrl Click (Command Click on OS X)** shortcut on a reference to display its definition.

Refactoring Actions

The following refactoring actions are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

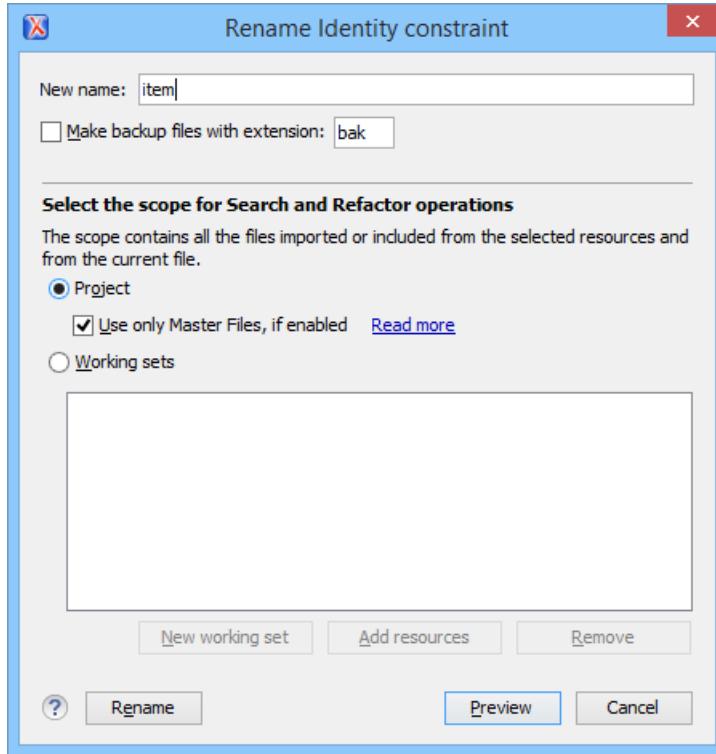


Figure 179: Rename Identity Constraint Dialog Box

Searching and Refactoring Operations Scope in WSDL Documents

The **scope** is a collection of documents that define the context of a search and refactor operation. To control it you can use the  **Change scope** operation, available in the Quick Assist action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.

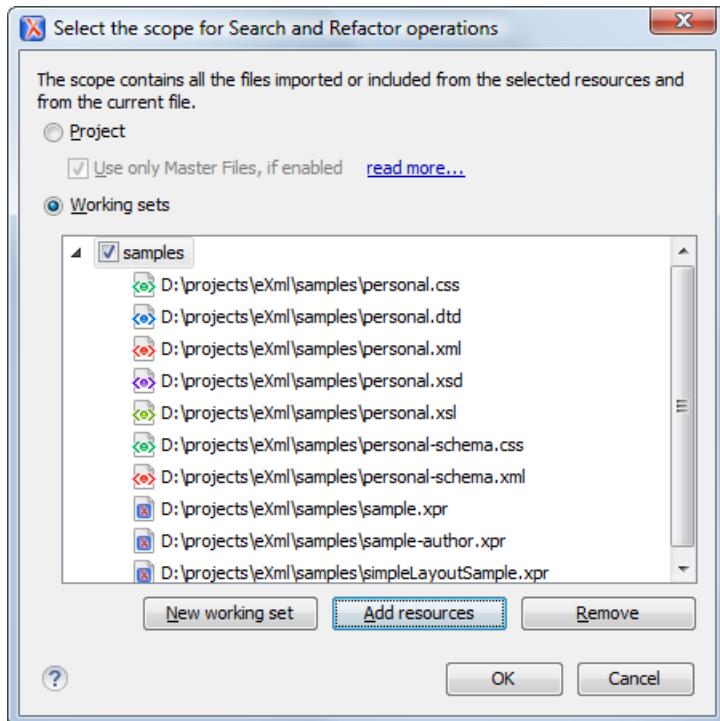


Figure 180: Change Scope Dialog

The scope you define is applied to all future search and refactor operations until you modify it. Contextual menu actions allow you to add or delete files, folders, and other resources to the working set structure.

WSDL Resource Hierarchy/Dependencies View in WSDL Documents

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a WSDL resource. To open this view, go to **Window > Show View > Resource Hierarchy/Dependencies**.

 **Note:** The hierarchy of a WSDL resource includes the hierarchy of imported XML Schema resources. The dependencies of an XML Schema resource present the WSDL documents that import the schema.

To view the hierarchy of a WSDL document, select the document in the project view and choose **Resource Hierarchy** from the contextual menu.

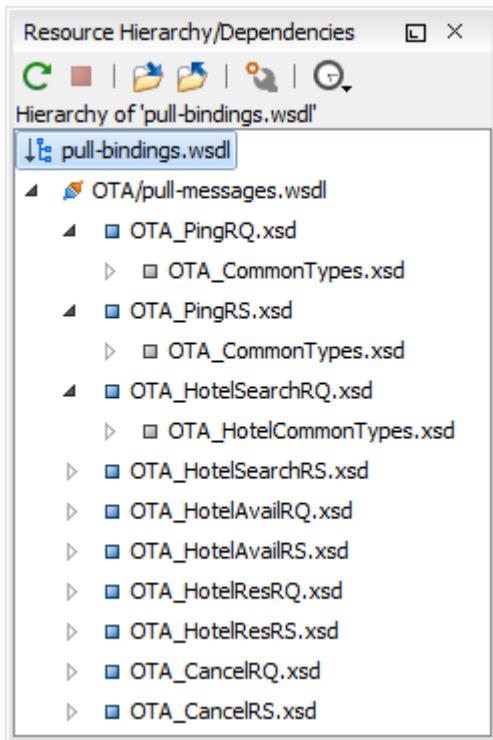


Figure 181: Resource Hierarchy/Dependencies View

If you want to see the dependencies of a WSDL document, select the document in the project view and choose **Resource Dependencies** from the contextual menu.

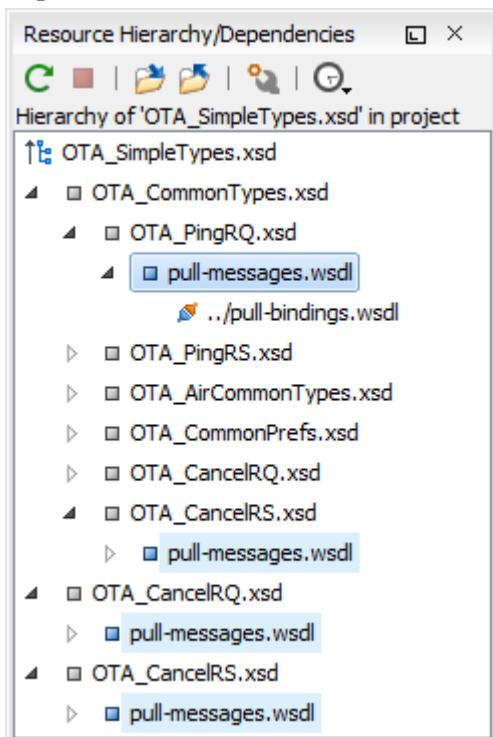


Figure 182: Resource Hierarchy/Dependencies View

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the  **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.



Tip: When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .



Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming WSDL Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Component Dependencies View in WSDL Documents

The **Component Dependencies** view allows you to view the dependencies for a selected WSDL component. To open the **Component Dependencies** view, go to **Window > Show View > Component Dependencies**.

To view the dependencies of an WSDL component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. This action is available for all WSDL components (messages, port types, operations, bindings and so on).

 **Note:** If you search for dependencies of XML Schema elements, the **Component Dependencies** view presents the references from WSDL documents.

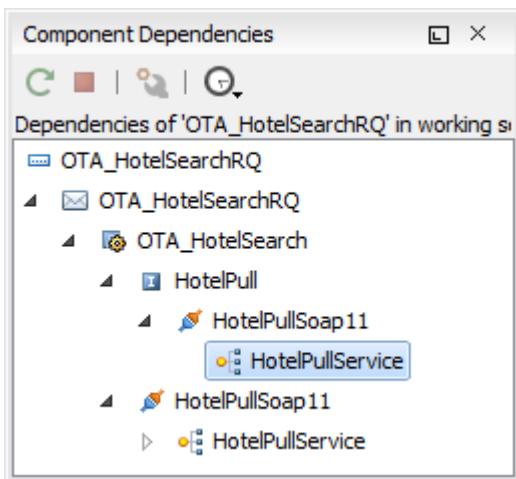


Figure 183: Component Dependencies View

The following action are available in the toolbar of the **Component Dependencies** view:

 **Refresh**

Refreshes the dependencies structure.

 **Stop**

Stops the dependencies computing.

 **Configure**

Allows you to configure a *search scope* to compute the dependencies structure. You can decide to use the defined scope for future operations automatically, by checking the corresponding check box.

 **History**

Allows you to repeat a previous dependencies computation.

The following actions are available in the contextual menu of the **Component Dependencies** view:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Displays the definition of the current selected component in the dependencies tree.

- Tip:** If a component contains multiple references to another, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon .

Highlight Component Occurrences in WSDL Documents

When you position your mouse cursor over a component in a WSDL document, Oxygen XML Editor searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default behaviour of **Highlight Component Occurrences**, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. You can also trigger a search using the **Search > Search Occurrences in File ()** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Quick Assist Support in WSDL Documents

Quick Assist is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

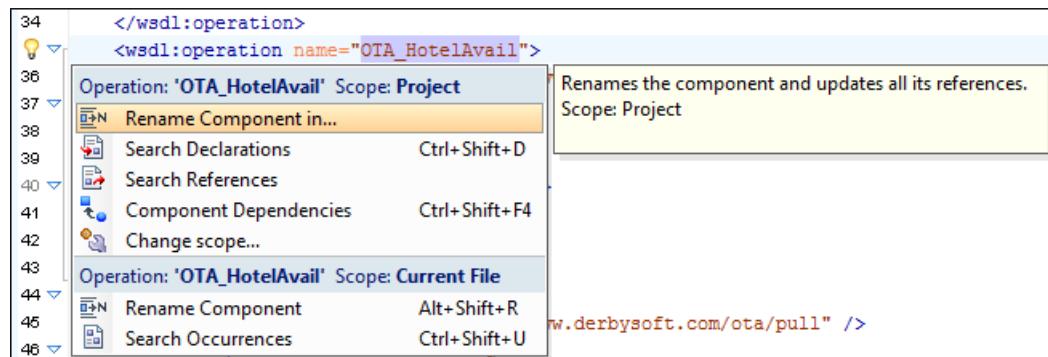


Figure 184: WSDL Quick Assist Support

The quick assist support offers direct access to the following actions:

Rename Component in...

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope...

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Generating Documentation for WSDL Documents

You can use Oxygen XML Editor to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the level of details to present for each of them. Also, the components are hyperlinked.

 **Note:** The WSDL documentation includes the XML Schema components that belong to the internal or imported XML schemas.

 **Note:** To obtain the documentation in a custom format, *use custom stylesheets*.

To generate documentation for a WSDL document, select **WSDL Documentation...** from the **Tools > Generate Documentation** menu or from the **Generate Documentation** submenu in the contextual menu of the **Project** view.

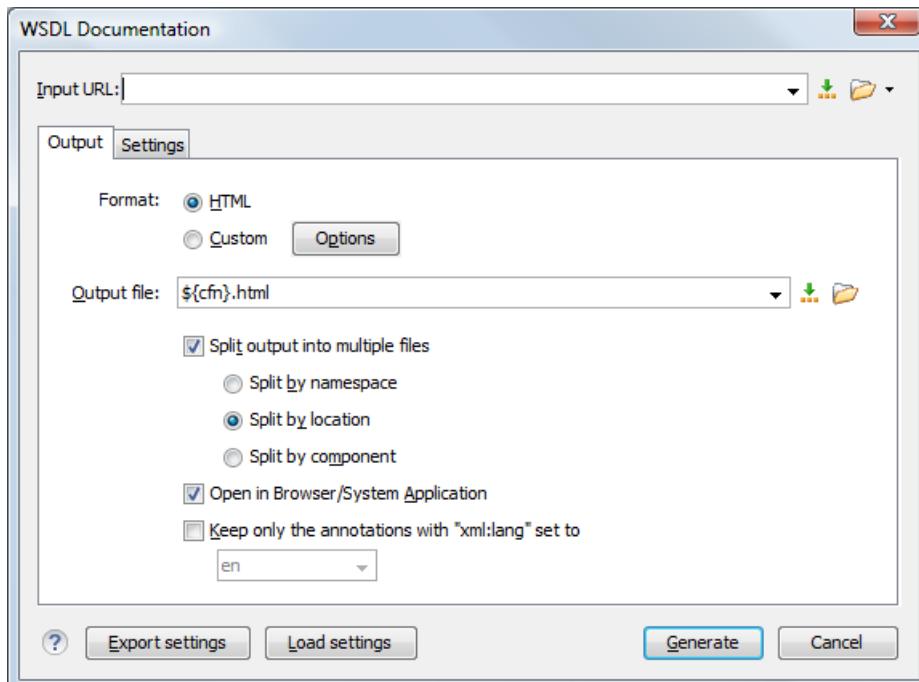


Figure 185: The Output Panel of the WSDL Documentation Dialog Box

The **Input URL** field of the dialog panel must contain the full path to the WSDL document that you want to generate documentation for. The WSDL document can be located either local or remote. You can also specify the path to the WSDL document using editor variables.

You can split the output into multiple files using different criteria. For large WSDL documents, choosing a different split criterion may generate smaller output files providing a faster documentation browsing.

The available split criteria are:

- By location - each output file contains the components from the same WSDL document.
- By namespace - each output file contains information about components with the same namespace.

- By component - each output file contains information about one WSDL or XML Schema component.

To export the settings of the **WSDL Documentation** dialog to an XML file, click the **Export settings** button. With the exported settings file, you can generate the same *documentation from the command-line interface*.

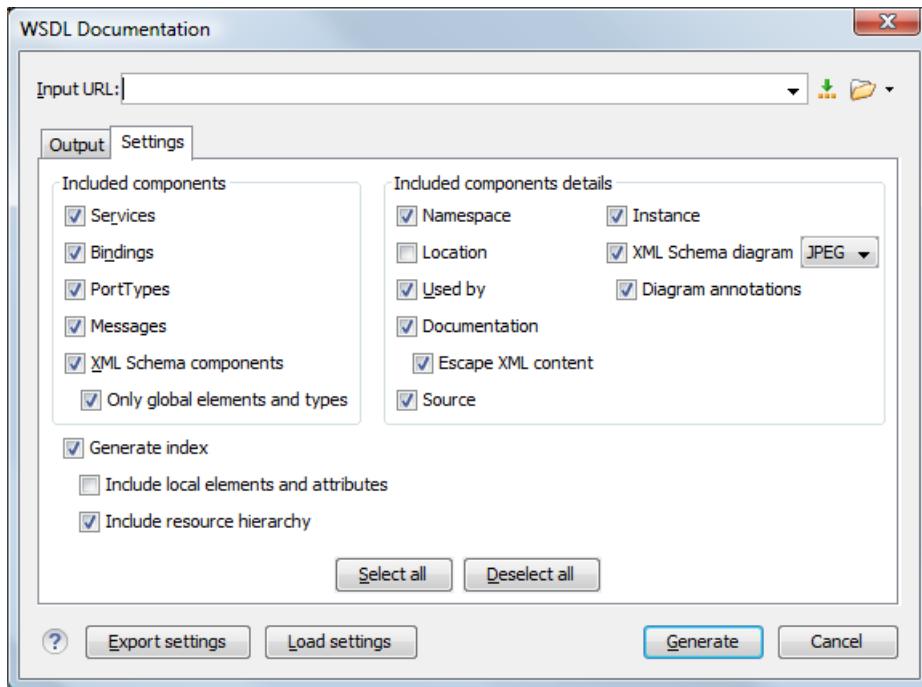


Figure 186: The Settings Panel of the WSDL Documentation Dialog

When you generate documentation for WSDL documents, you can choose what components (services, bindings, messages and others) and details (namespace, location, instance and others) to include in the documentation:

- **Components**

- **Services** - specifies whether the generated documentation includes the WSDL services.
- **Bindings** - specifies whether the generated documentation includes the WSDL bindings.
- **Port Types** - specifies whether the generated documentation includes the WSDL port types.
- **Messages** - specifies whether the generated documentation includes the WSDL messages.
- **XML Schema Components** specifies whether the generated documentation includes the XML Schema components.
- **Only global elements and types** - specifies whether the generated documentation includes only global elements and types.

- **Details**

- **Namespace** - presents the namespace information for WSDL or XML Schema components.
- **Location** - presents the location information for each WSDL or XML Schema component.
- **Used by** - presents the list of components that reference the current one.
- **Documentation** - presents the component documentation. In case you choose **Escape XML Content**, the XML tags are presented in the documentation.
- **Source** - presents the XML fragment that defines the current component.
- **Instance** - generates a sample XML instance for the current component.



Note: This option applies to the XML Schema components only.

- **XML Schema Diagram** - Displays the diagram for each XML Schema component. You can choose the image format (JPEG, PNG, GIF, SVG) to use for the diagram section.

- **Diagram annotations** - specifies whether the annotations of the components presented in the diagram sections are included.

Generating Documentation for WSDL Documents in HTML Format

The default format of the generated WSDL documentation is HTML.

Main WSDL stockQuoteService.wsdl

Name	StockQuote
Namespace	http://example.com/stockquote/service

Service tns:StockQuoteService

Namespace	http://example.com/stockquote/service
Documentation	Provides the last trade price for a stock.
Ports	StockQuotePort
Source	<pre> <wsdl:service name="StockQuoteService"> <wsdl:documentation>Provides the last trade price for a stock.</wsdl:documentation> <wsdl:port binding="tns:StockQuoteSoap" name="StockQuotePort"> <soap:address location="http://example.com /stockquote"/> </wsdl:port> </wsdl:service></pre>

Showing:

- Ports
- Operations
- Documentation
- Source
- Used by

Figure 187: WSDL Documentation in HTML Format

By default, the documentation of each component is presented to the right side. Each component is displayed in a separate section. The title of the section is composed of the component type and the component name. The component information (namespace, documentation and so on) is presented in a tabular form. The left side of the output holds the table of contents. The table of contents is divided in two tabs: **Components** and **Resource Hierarchy**.

The **Components** tab allows you to group the contents by namespace, location, or component type. The WSDL components from each group are sorted alphabetically. The **Resource Hierarchy** tab displays the dependencies between WSDL and XML Schema modules in a tree like fashion. The root of the tree is the WSDL document that you generate documentation for.

If you split the output in multiple files, the table of contents is displayed in the left frame. The contents are grouped using the same criteria as the split.

After the documentation is generated, you can collapse details for some WSDL components using the **Showing** view.

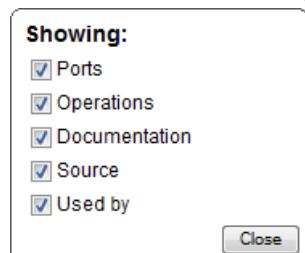


Figure 188: The Showing View

Generating Documentation for WSDL Documents in a Custom Format

To obtain the default HTML documentation output from a WSDL document, Oxygen XML Editor uses an intermediary XML document to which it applies an XSLT stylesheet. To create a custom output from your WSDL document, edit this XSLT stylesheet or write your own one.

-  **Note:** The wsdlDocHtml.xsl stylesheet used to obtain the HTML documentation is located in the installation folder of Oxygen XML Editor, in the [OXYGEN_DIR]/frameworks/wsdl_documentation/xsl folder.
-  **Note:** The intermediary XML document complies with the wsdlDocSchema.xsd XML Schema. This schema is located in the installation folder of Oxygen XML Editor, in the [OXYGEN_DIR]/frameworks/wsdl_documentation folder.

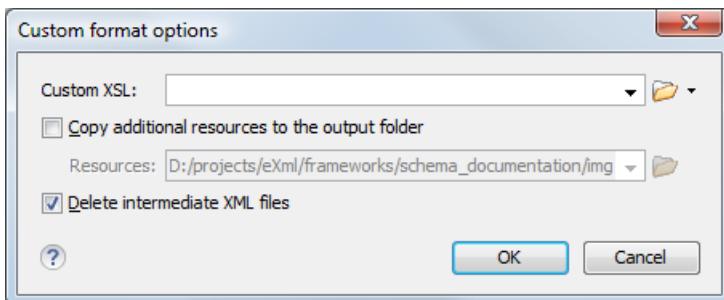


Figure 189: The Custom Format Options Dialog

When using a custom format, you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating Documentation for WSDL Documents from the Command Line

To generate documentation from a WSDL document from the command line, open the **WSDL Documentation** dialog and click **Export settings**. Using the exported settings file you can generate the same documentation from the command line by running the following scripts:

- wsdlDocumentation.bat on Windows.
- wsdlDocumentation.sh on Unix / Linux.
- wsdlDocumentationMac.sh on Mac OS X.

The scripts are located in the installation folder of Oxygen XML Editor. You can integrate the scripts in an external batch process launched from the command-line interface.

WSDL SOAP Analyzer

After you edit and validate your Web service descriptor against a mix of the XML Schemas for WSDL and SOAP, it is easy to check if the defined SOAP messages are accepted by the remote Web Services server using Oxygen XML Editor's **WSDL SOAP Analyser** integrated tool.

Composing a SOAP Request

WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

Oxygen XML Editor provides two ways of testing, one for the currently edited WSDL document and another for the remote WSDL documents that are published on a web server. To open the **WSDL SOAP Analyser** tool for the currently edited WSDL document do one of the following:

- Click the  **WSDL SOAP Analyser** toolbar button.
- Go to **Document > Tools > WSDL SOAP Analyser**.
- Go to **Open with > WSDL SOAP Analyser** in the contextual menu of the **Project** view.

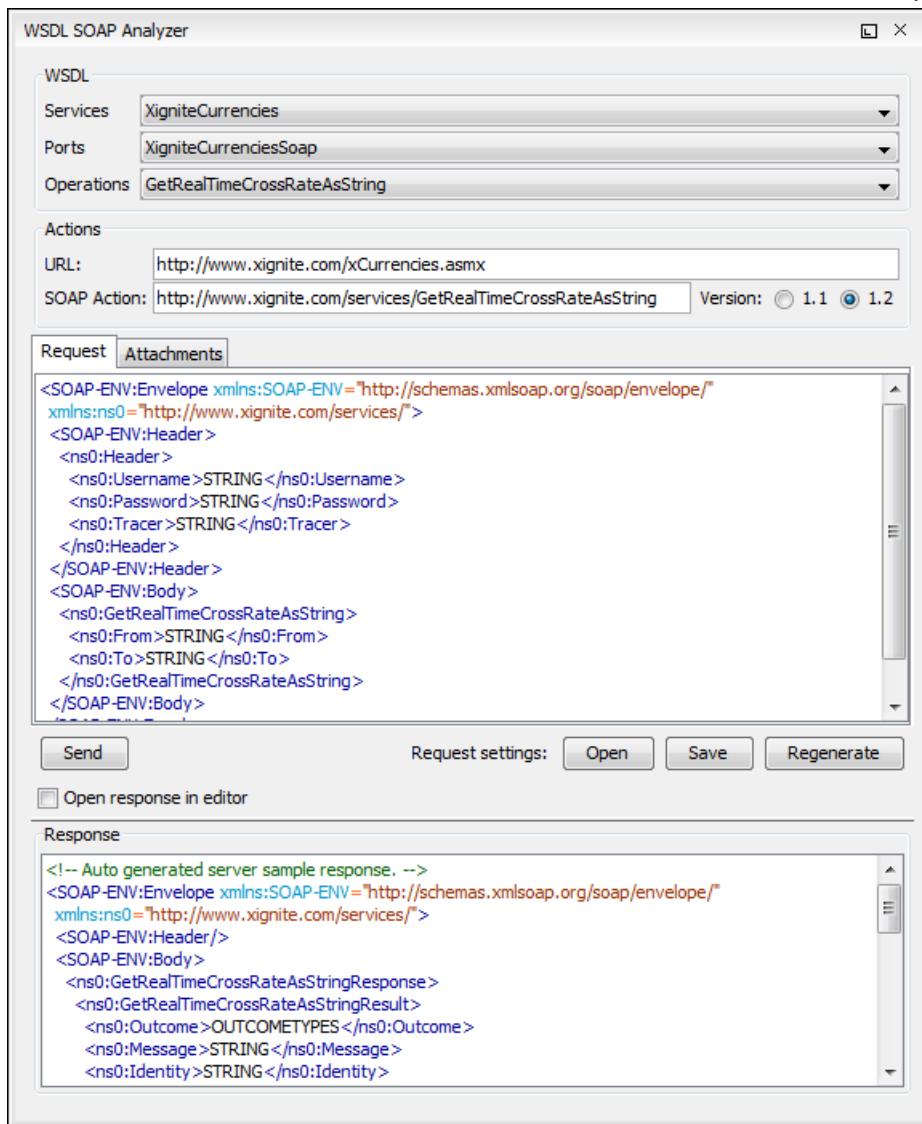


Figure 190: WSDL SOAP Analyser

This dialog box contains a SOAP analyser and sender for Web Services Description Language file types. The analyser fields are:

- **Services** - The list of services defined by the WSDL file.
- **Ports** - The ports for the selected service.
- **Operations** - The list of available operations for the selected service.
- **Action URL** - Shows the script that serves the operation.
- **SOAP Action** - Identifies the action performed by the script.
- **Version** - Choose between 1.1 and 1.2. The SOAP version is selected automatically depending on the selected port.
- **Request Editor** - It allows you to compose the web service request. When an action is selected, Oxygen XML Editor tries to generate as much content as possible for the SOAP request. The envelope of the SOAP request has the correct namespace for the selected SOAP version, that is <http://schemas.xmlsoap.org/soap/envelope/> for SOAP 1.1 or <http://www.w3.org/2003/05/soap-envelope> for SOAP 1.2. Usually you just have to change few values in order for the request to be valid. The content completion assistant is available for this editor and is driven by the schema that defines the type of the current message. While selecting different operations, Oxygen XML Editor remembers the modified request for each one. You can press the **Regenerate** button in order to overwrite your modifications for the current request with the initial generated content.
- **Attachments List** - You can define a list of file URLs to be attached to the request.

- **Response Area** - Initially it displays an auto generated server sample response so you can have an idea about how the response looks like. After pressing the **Send** button, it presents the message received from the server in response to the Web Service request. It may show also error messages. In case the response message contains attachments, Oxygen XML Editor prompts you to save them, then tries to open them with the associated system application.
- **Errors List** - There may be situations in which the WSDL file is respecting the WSDL XML Schema, but it fails to be valid for example in the case of a message that is defined by means of an element that is not found in the types section of the WSDL. In such a case, the errors are listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog box is displayed when Oxygen XML Editor is connecting to the server.

The testing of a WSDL file is straight-forward: click the WSDL analysis button, then select the service, the port, and the operation. The editor generates the skeleton for the SOAP request. You can edit the request, eventually attach files to it and send it to the server. Watch the server response in the response area. You can find more details in the [Testing Remote WSDL Files](#) section.



Note: SOAP requests and responses are automatically validated in the **WSDL SOAP Analyser** using the XML Schemas specified in the WSDL file.

Once defined, a request derived from a Web Service descriptor can be saved with the **Save** button to a Web Service SOAP Call (WSSC) file for later reuse. In this way, you save time in configuring the URLs and parameters.

You can open the result of a Web Service call in an editor panel using the **Open** button.

Testing Remote WSDL Files

To open and test a remote WSDL file the steps are the following:

1. Go to menu **Tools > WSDL SOAP Analyser ...**.
2. On the **WSDL File** tab enter the URL of the remote WSDL file.

You enter the URL:

- by typing
- by browsing the local file system
- by browsing a remote file system
- by browsing [a UDDI Registry](#)

3. Press the **OK** button.

This will open the **WSDL SOAP Analyser** tool. In the **Saved SOAP Request** tab you can open directly a previously saved Web Service SOAP Call (WSSC) file thus skipping the analysis phase.

The UDDI Registry Browser

Pressing the button in the **WSDL File Opener** dialog (menu **Tools > WSDL SOAP Analyzer**) opens the **UDDI Registry Browser** dialog.

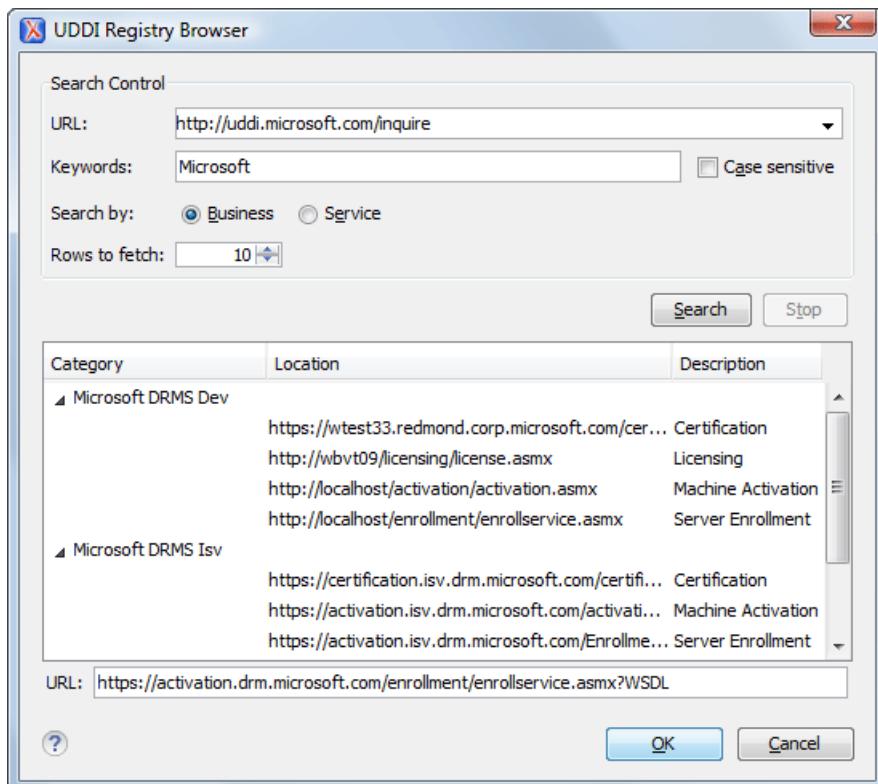


Figure 191: UDDI Registry Browser dialog

The fields of the dialog are the following:

- **URL** - Type the URL of an UDDI registry or choose one from the default list.
- **Keywords** - Enter the string you want to be used when searching the selected UDDI registry for available Web services.
- **Rows to fetch** - The maximum number of rows to be displayed in the result list.
- **Search by** - You can choose to search either by company or by provided service.
- **Case sensitive** - When checked, the search takes into account the keyword case.
- **Search** - The WSDL files that matched the search criteria are added in the result list.

When you select a WSDL from the list and click the **OK** button, the **UDDI Registry Browser** dialog is closed and you are returned to the WSDL File Opener dialog.

Editing CSS Stylesheets

This section explains the features of the editor for CSS stylesheets and how these features should be used.

Validating CSS Stylesheets

Oxygen XML Editor includes a built-in *CSS Validator*, integrated with general validation support. This makes the [usual validation features](#) for presenting errors also available for CSS stylesheets.

When you edit a CSS document, you can access the *CSS validator options* by selecting **Validation options** from the **Document > Validate** menu.

The CSS properties accepted by the validator are those included in the current CSS profile that is selected in [the CSS validation preferences](#). The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties plus the [CSS extensions specific for Oxygen](#) that can be used in *Author mode*. That means all Oxygen specific extensions are accepted in a CSS stylesheet by [the built-in CSS validator](#) when this profile is selected.

Specify Custom CSS Properties

Lists the steps required for specifying custom CSS properties.

To specify custom CSS properties, follow these steps:

1. Create a file named `CustomProperties.xml` that has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<css_keywords
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.oxygenxml.com/ns/css http://www.oxygenxml.com/ns/css/CssProperties.xsd"
    xmlns="http://www.oxygenxml.com/ns/css">
    <property name="custom">
        <summary>Description for custom property.</summary>
        <value name="customValue"/>
        <value name="anotherCustomValue"/>
    </property>
</css_keywords>
```

2. Go to your desktop and create the `builtin/css-validator/` folder structure.
3. Press and hold **Shift** and right-click on your desktop. From the contextual menu, select **Open Command Window Here**.
4. In the command line, run the `jar cvf custom_props.jar builtin/` command.
The `custom_props.jar` file is created.
5. Go to `[OXYGEN_DIR]/lib` and create the `endorsed` folder. Copy the `custom_props.jar` file to `[OXYGEN_DIR]/lib/endorsed`.

Content Completion in CSS Stylesheets

A **Content Completion Assistant**, similar to *the one available for XML documents* offers the CSS properties and the values available for each property. It is activated with the **Ctrl Space (Command Space on OS X)** shortcut and is context-sensitive when invoked for the value of a property. The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into CSS stylesheets. The code templates that are proposed include form controls, actions, and **Author** mode operations.

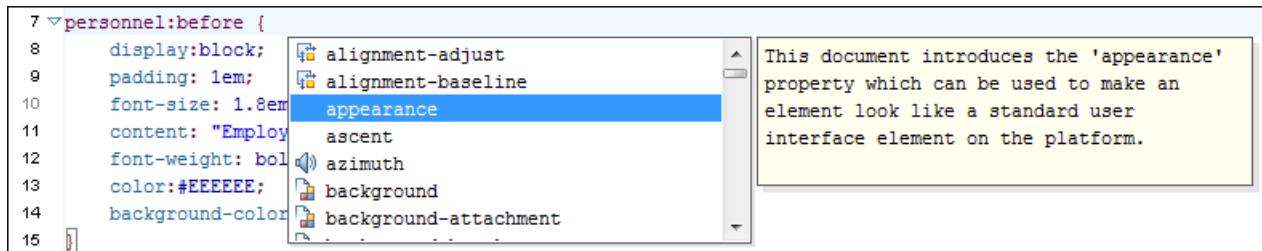


Figure 192: Content Completion in CSS Stylesheets

The properties and values available are dependent on the CSS Profile selected in the *CSS preferences*. The CSS 2.1 set of properties and property values is used for most of the profiles. However, with CSS 1 and CSS 3 specific proposal sets are used.

The profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties plus the *CSS extensions specific for Oxygen* that can be used in *Author mode*.

CSS Outline View

The **CSS Outline** view presents the import declarations for other CSS stylesheet files and all the selectors defined in the current CSS document. The selector entries can be presented as follows:

- in the order they appear in the document
- sorted by the element name used in the selector
- sorted by the entire selector string representation

You can synchronize the selection in the **Outline** view with the caret moves or changes you make in the stylesheet document. When you select an entry from the **Outline** view, Oxygen XML Editor highlights the corresponding import or selector in the CSS editor.

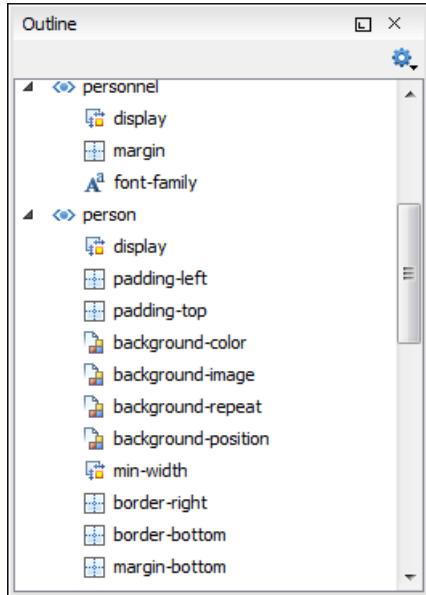


Figure 193: CSS Outline View

The selectors presented in this view can be found quickly using the key search field. When you press a sequence of character keys while the focus is in the view, the first selector that starts with that sequence is selected automatically.

Folding in CSS Stylesheets

In a large CSS stylesheet document, some styles can be collapsed so that only the styles that are needed remain in focus. The same *folding features available for XML documents* are also available in CSS stylesheets.

 **Note:** To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.

Formatting and Indenting CSS Stylesheets (Pretty Print)

If the edited CSS stylesheet becomes unreadable because of the bad alignment of the text lines, the *format and indent operation available for XML documents* is also available for CSS stylesheets. It works in the same way as for XML documents and is available as the same menu and toolbar action.

Minifying CSS Stylesheets

Minification (or *compression*) of a CSS document is the practice of removing unnecessary code without affecting the functionality of the stylesheet.

To minify a CSS, invoke the contextual menu anywhere in the edited document and choose the **Minify CSS...** action. Oxygen XML Editor opens a dialog box that allows you to:

- Set the location of the resulting CSS.
- Place each style rule on a new line.

After pressing **OK**, Oxygen XML Editor performs the following actions:

- All spaces are normalized (all leading and trailing spaces are removed, while sequences of white spaces are replaced with single space characters).
- All comments are removed.



Note: The CSS minifier relies heavily upon the W3C CSS specification. If the content of the CSS file you are trying to minify does not conform with the specifications, an error dialog box will be displayed, listing all errors encountered during the processing.

The resulting CSS stylesheet gains a lot in terms of execution performance, but loses in terms of readability. The source CSS document is left unaffected.



Note: To restore the readability of a minified CSS, invoke the **Format and Indent** action from the **Document > Source** menu, the **Source** submenu from the contextual menu, or **Source** toolbar. However, this action will not recover any of the deleted comments.

Other CSS Editing Actions

The CSS editor type offers a reduced version of *the popup menu available in the XML editor*. Only *the split actions*, *the folding actions*, *the edit actions* and a part of *the source actions* (only the actions **To lower case**, **To upper case**, **Capitalize lines**) are available.

Editing LESS CSS Stylesheets

Oxygen XML Editor provides support for stylesheets coded with the LESS dynamic stylesheet language. LESS extends the CSS language by adding features that allow mechanisms such as *variables*, *nesting*, *mixins*, *operators*, and *functions*. Oxygen XML Editor offers additional LESS features that include:

- Open LESS files - the LESS extension is recognized and thus can be opened by the editor
- Validation - presents errors in LESS files
- Content completion - offers properties and the values available for each property
- Compile to CSS - options are available to compile LESS files to CSS



Note: Oxygen XML Editor also supports syntax highlighting in LESS files, although there may be some limitations in supporting all the LESS constructs.

For more information about LESS go to <http://lesscss.org/>.

Validating LESS Stylesheets

Oxygen XML Editor includes a built-in *LESS CSS Validator*, integrated with general validation support. The *usual validation features* for presenting errors also available for LESS stylesheets.

Oxygen XML Editor provides three validation methods:

- Automatic validation as you type - marks validation errors in the document as you are editing.
- Validation upon request, by pressing the **Validate** button from the **Validation** toolbar drop-down list. An error list is presented in the message panel at the bottom of the editor.
- Validation scenarios, by selecting **Configure Validation Scenario(s)...** from the **Validation** toolbar drop-down list. Errors are presented in the message panel at the bottom of the editor. This is useful when you need to validate the current file as part of a larger LESS import hierarchy (for instance, you may change the URL of the file to validate to the root of the hierarchy).

Content Completion in LESS Stylesheets

A **Content Completion Assistant** offers the LESS properties and the values available for each property. It is activated with the Ctrl Space (Command Space on OS X) shortcut and is context-sensitive when invoked for the value of a property in a LESS file. The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into LESS stylesheets. The code templates that are proposed include form controls, actions, and **Author** mode operations.

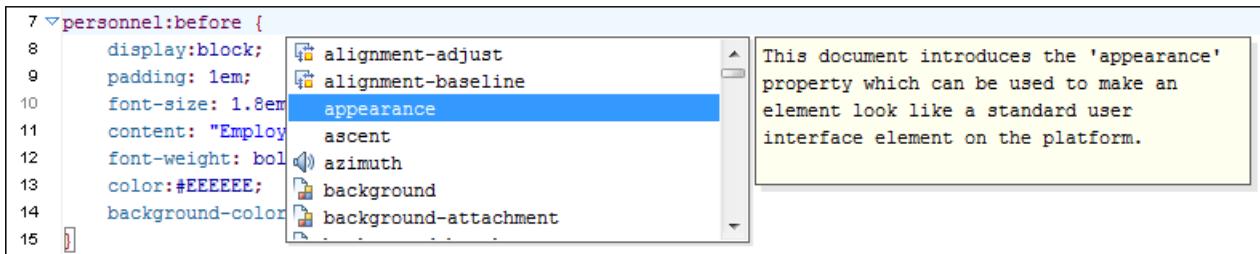


Figure 194: Content Completion in LESS Stylesheets

The properties and values available are dependent on the CSS Profile selected in the [CSS preferences](#).

Compiling LESS Stylesheets to CSS

When editing LESS files, you can compile the files into CSS. Oxygen XML Editor provides both manual and automatic options to compile LESS stylesheets into CSS.

You have two options for compiling LESS files to CSS:

1. Use the contextual menu in a LESS file and select **Compile to CSS (Ctrl Shift C (Command Shift C on OS X))**.
2. Enable the option **Automatically compile LESS to CSS when saving** in the settings. To do so, [open the Preferences dialog box](#) and go to **Editor > Open > Save > Save hooks**. If enabled, when you save a LESS file it will automatically be compiled to CSS (this option is disabled by default).

Important: If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

Editing Relax NG Schemas

Oxygen XML Editor provides a special type of editor for Relax NG schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the [standard outline](#) mode and the [components](#) mode.

Editing Relax NG Schema in the Master Files Context

Smaller interrelated modules that define a complex Relax NG Schema cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, an element defined in a main schema document is not visible when you edit an included module. Oxygen XML Editor provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You can set a main Relax NG document either using the [master files support from the Project view](#), or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- correct validation of a module in the context of a larger schema structure;
- **Content Completion Assistant** displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one;
- the **Outline** displays the components collected from the entire schema structure;

Relax NG Schema Diagram

This section explains how to use the graphical diagram of a Relax NG schema.

Introduction

Oxygen XML Editor provides a simple, expressive, and easy to read **Schema Diagram** view for Relax NG schemas.

With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, or BMP images. It helps both schema authors in developing the schema and content authors who are using the schema to understand it.

Oxygen XML Editor is the only XML editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- the changes you make in the Editor are immediately visible in the Diagram (no background parsing);
- changing the selected element in the diagram selects the underlying code in the source editor.

Full Model View

When you create a new schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The **Diagram** view has two tabbed panes offering a **Full Model View** and a **Logical Model View**.

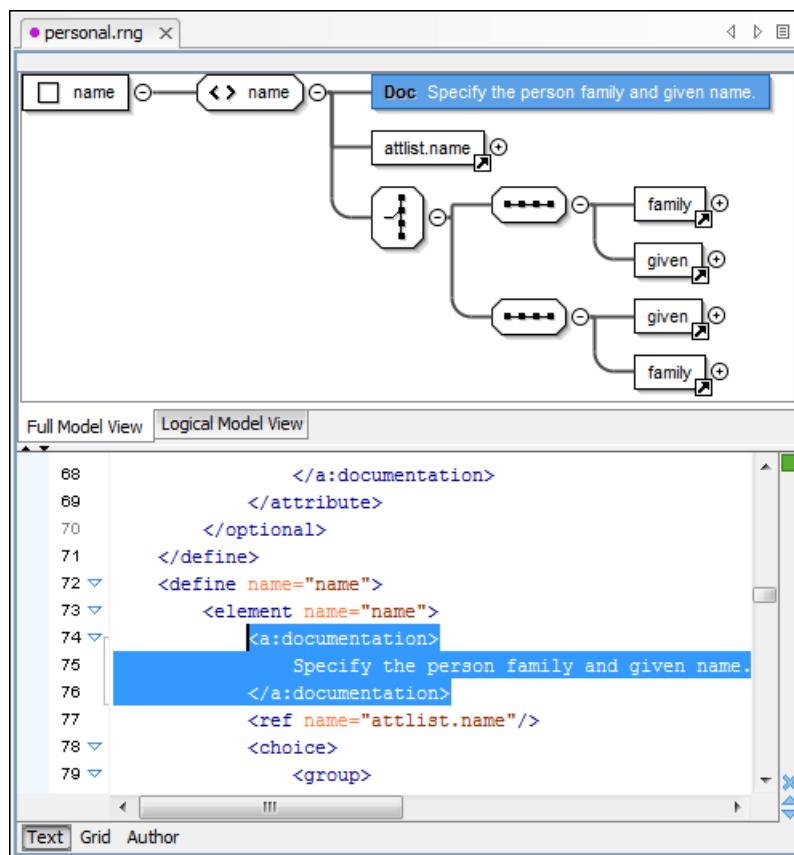


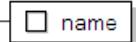
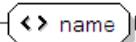
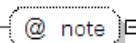
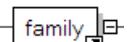
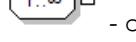
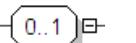
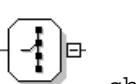
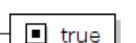
Figure 195: Relax NG Schema Editor - Full Model View

The following references can be expanded in place: patterns, includes, and external references. This expansion mechanism, coupled with the synchronization support, makes the schema navigation easy.

All the element and attribute names are editable: double-click any name to start editing it.

Symbols Used in the Schema Diagram

The **Full Model View** renders all the Relax NG Schema patterns with intuitive symbols:

-  - define pattern with the name attribute set to the value shown inside the rectangle (in this example name).
-  - define pattern with the combine attribute set to interleave and the name attribute set to the value shown inside the rectangle (in this example attlist.person).
-  - define pattern with the combine attribute set to choice and the name attribute set to the value shown inside the rectangle (in this example attlist.person).
-  - element pattern with the name attribute set to the value shown inside the rectangle (in this example name).
-  - attribute pattern with the name attribute set to the value shown inside the rectangle (in this case note).
-  - ref pattern with the name attribute set to the value shown inside the rectangle (in this case family).
-  - oneOrMore pattern.
-  - zeroOrMore pattern.
-  - optional pattern.
-  - choice pattern.
-  - value pattern, used for example inside a choice pattern.
-  - group pattern.
-  - pattern from the Relax NG Annotations namespace (<http://relaxng.org/ns/compatibility/annotations/1.0>) which is treated as a documentation element in a Relax NG schema.
-  - text pattern.
-  - empty pattern.

Logical Model View

The **Logical Model View** presents the compiled schema which is a single pattern. The patterns that form the element content are defined as top level patterns with generated names. These names are generated depending of the elements name class.

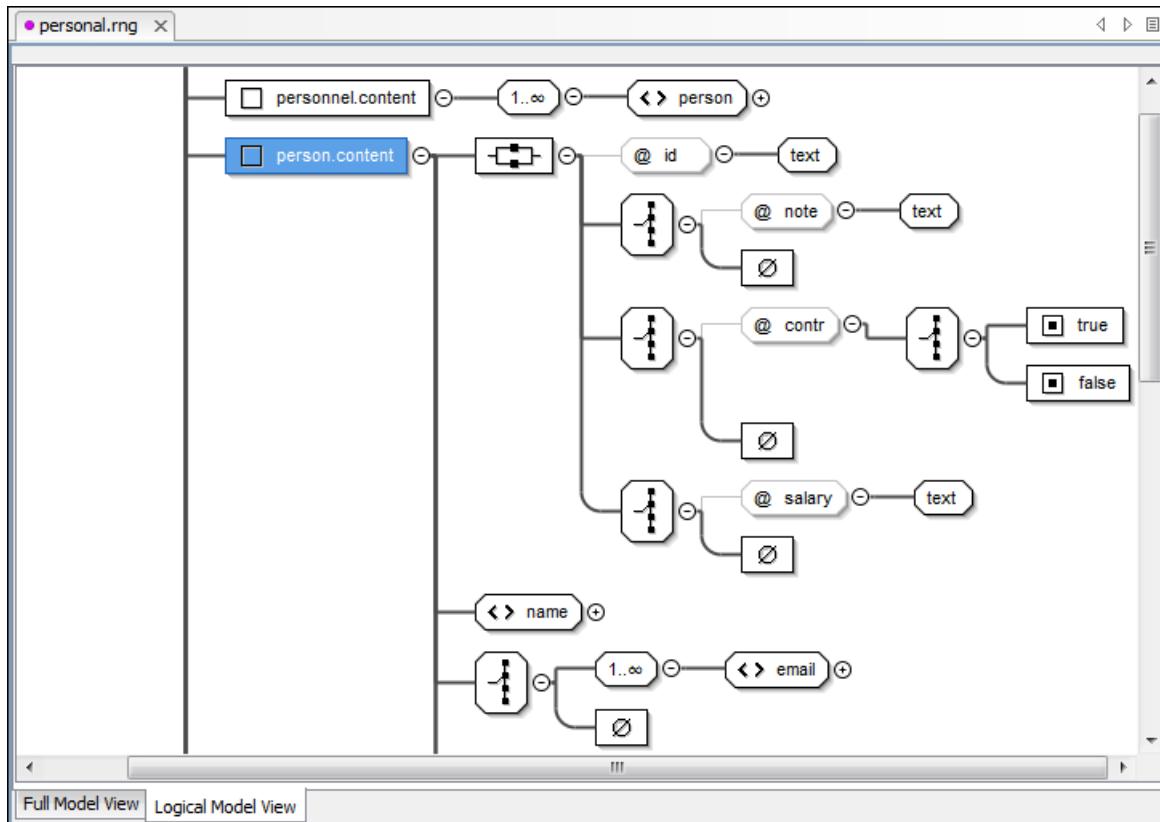


Figure 196: Logical Model View for a Relax NG Schema

Actions Available in the Diagram View

The contextual menu offers the following actions:

Append child

Appends a child to the selected component.

Insert Before

Inserts a component before the selected component.

Insert After

Inserts a component after the selected component.

Edit attributes

Edits the attributes of the selected component.

Remove

Removes the selected component.

Show only the selected component

Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.

Show Annotations

Depending on its state (selected/not selected), the documentation nodes are shown or hidden.

Auto expand to references

This option controls how the schema diagram is automatically expanded. If you select it and then edit a top-level element or you make a refresh, the diagram is expanded until it reaches referenced components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.

Collapse Children

Collapses the children of the selected view.

Expand Children

Expands the children of the selected view.

Print Selection...

Prints the selected view.

Save as Image...

Saves the current selection as JPEG, BMP, SVG or PNG image.

Refresh

Refreshes the schema diagram according to the changes in your code. They represent changes in your imported documents or changes that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

Relax NG Outline View

The Relax NG **Outline** view presents a list with the patterns that appear in the diagram in both the **Full Model View** and **Logical Model View** cases. It allows a quick access to a component by name. By default it is displayed on screen. If you closed the **Outline** view you can reopen it from menu **Window > Show View > Outline**. You can switch between the Relax NG patterns version and the *standard XML version* of the view by pressing the **Show components/Show XML structure** button.

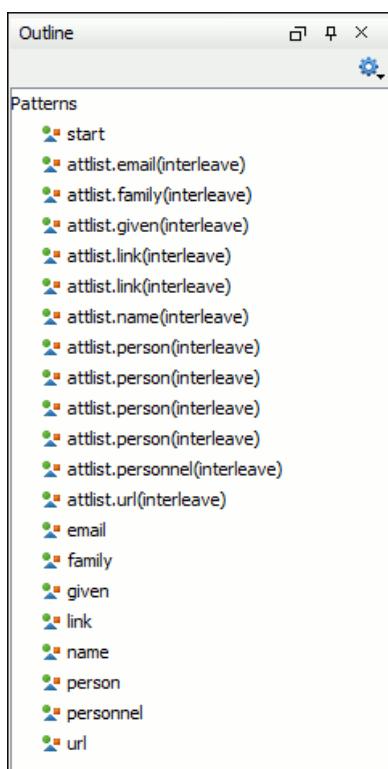


Figure 197: Relax NG Outline View

The tree shows the XML structure or the define patterns collected from the current document. By default, the **Outline** view presents the define patterns.

When the XML elements are displayed, the following actions are available in the **Settings** menu on the Outline view's toolbar:

 **Selection update on caret move**

Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram will be also selected in the **Outline** view.

 **Show components**

Shows the define patterns collected from the current document.

 **Flat presentation mode of the filtered results**

When active, the application flattens the filtered result elements to a single level.

 **Show comments and processing instructions**

Show/hide comments and processing instructions in the **Outline** view.

 **Show element name**

Show/hide element name.

 **Show text**

Show/hide additional text content for the displayed elements.

 **Show attributes**

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).

 **Configure displayed attributes**

Displays the [XML Structured Outline preferences page](#).

When components are displayed, the following action is available in the **Settings** menu on the Outline view's toolbar:

 **Show XML structure**

Shows the XML structure of the current document.

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Relax NG Editor Specific Actions

The list of actions specific for the Relax NG (full syntax) editor is:

- **Document > Schema > Show Definition** (also available on the contextual menu of the editor panel) - Moves the cursor to the definition of the current element in this Relax NG (full syntax) schema. You can use the [Ctrl Click](#) ([Command Click on OS X](#)) shortcut on a reference to display its definition.

Searching and Refactoring Actions in RNG Schemas

Search Actions

The following search actions can be applied on named *defines* and are available from the **Search** submenu in the contextual menu of the current editor or from the **Document > References** menu:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on named *defines* and are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

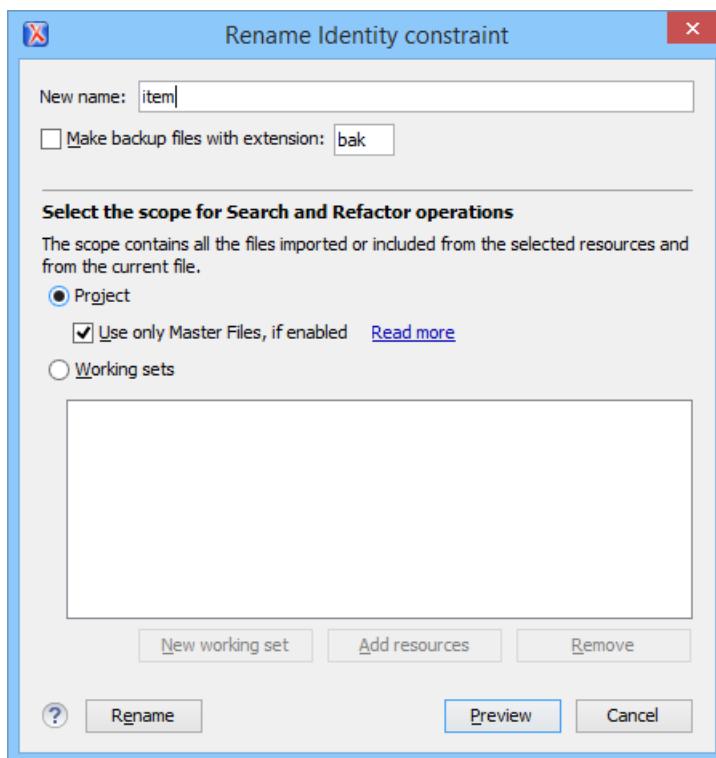


Figure 198: Rename Identity Constraint Dialog Box

RNG Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a schema. To open this view, go to **Window > Show View > Resource Hierarchy/Dependencies**.

If you want to see the hierarchy of a schema, select the desired schema in the project view and choose **Resource Hierarchy** from the contextual menu.

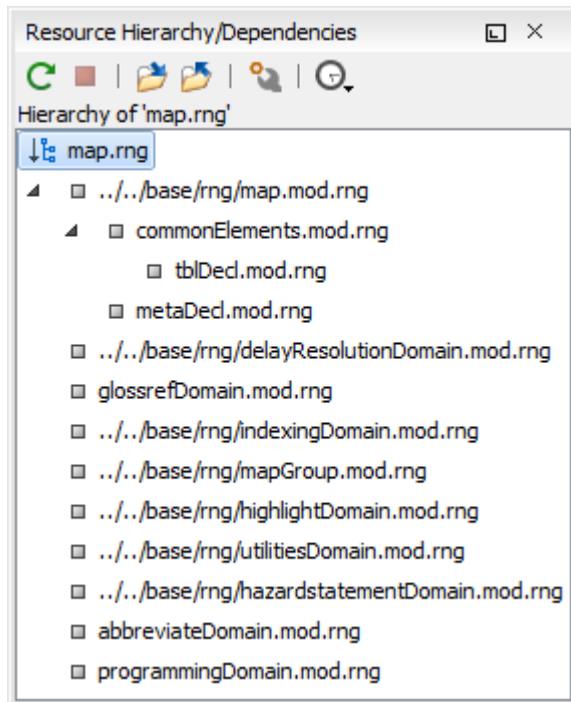


Figure 199: Resource Hierarchy/Dependencies View - hierarchy for `map.rng`

If you want to see the dependencies of a schema, select the desired schema in the project view and choose **Resource Dependencies** from the contextual menu.

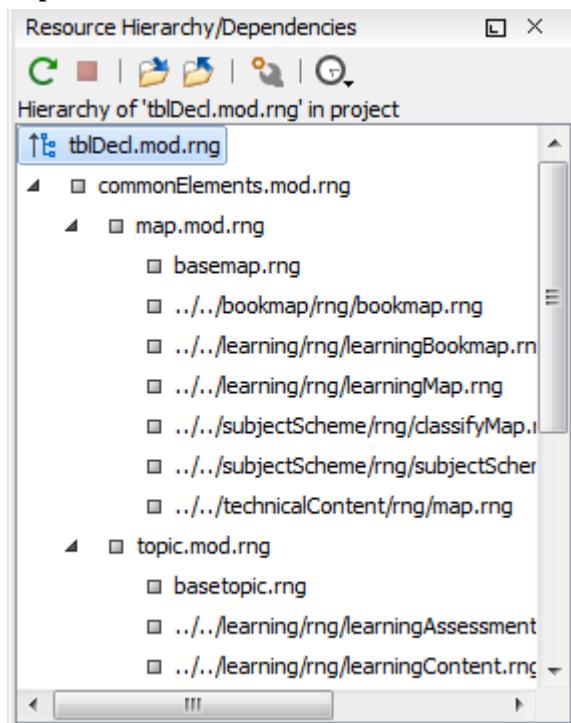


Figure 200: Resource Hierarchy/Dependencies View - Dependencies for `tblDecl.mod.rng`

The following actions are available in the **Resource Hierarchy/Dependencies** view:

C Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the  **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

 **Tip:** When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .



Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming RNG Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

 **Note:** Updating the references of a resource that is resolved through a catalog is not supported. Also, the update references operation is not supported in case the path to the renamed or moved resource contains entities.

Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected Relax NG component. You can open the view from **Window > Show View > Component Dependencies**.

If you want to see the dependencies of a RelaxNG component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named defines.

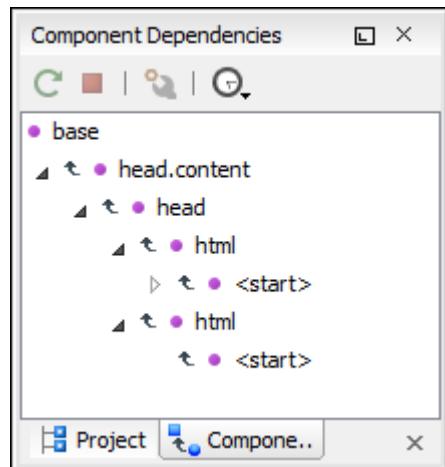


Figure 201: Component Dependencies View - Hierarchy for base.rng

In the Component Dependencies view you have several actions in the toolbar:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

History

Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

- Tip:** If a component contains multiple references to another components, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon .

RNG Quick Assist Support

The Quick Assist support improves the development work flow, offering fast access to the most commonly used actions when you edit XML Schema documents.

Quick Assist is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

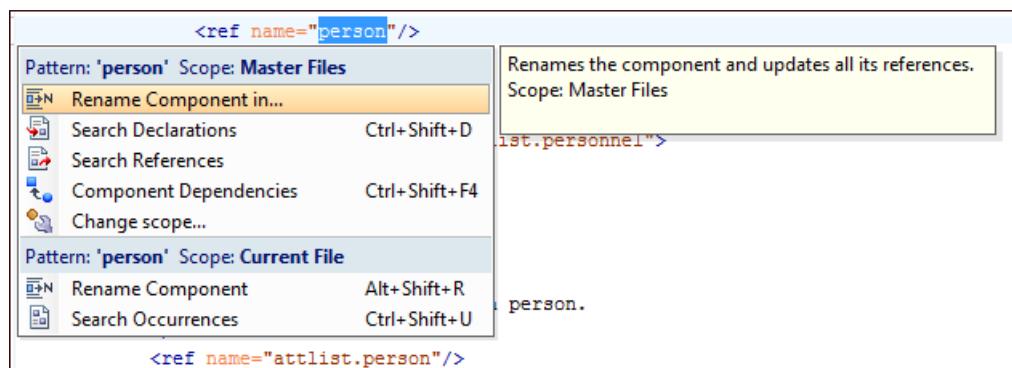


Figure 202: RNG Quick Assist Support

The quick assist support offers direct access to the following actions:

Rename Component in...

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope...

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Configuring a Custom Datatype Library for a RELAX NG Schema

A RELAX NG schema can declare a custom datatype library for the values of elements found in XML document instances. The datatype library must be developed in Java and it must implement the interface [specified on the www.thaiopensource.com website](http://www.thaiopensource.com/websrc/relaxng/doc/api/index.html).

The jar file containing the custom library and any other dependent jar file must be added to the classpath of the application, that is the jar files must be added to the folder [OXYGEN_DIR]/lib .

To load the custom library, restart Oxygen XML Editor.

Linking Between Development and Authoring

The **Author** mode is available on the Relax NG schema presenting the schema similar with the Relax NG compact syntax. It links to imported schemas and external references. Embedded Schematron is supported only in Relax NG schemas with XML syntax.

Editing NVDL Schemas

Some complex XML documents are composed by combining elements and attributes from different namespaces. More, the schemas that define these namespaces are not even developed in the same schema language. In such cases, it is difficult to specify in the document all the schemas which must be taken into account for validation of the XML document or for content completion. An NVDL (Namespace Validation Definition Language) schema can be used. This schema allows the application to combine and interleave multiple schemas of different types (W3C XML Schema, RELAX NG schema, Schematron schema) in the same XML document.

Oxygen XML Editor provides a special type of editor for NVDL schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the *standard outline* mode and the *components* mode.

NVDL Schema Diagram

This section explains how to use the graphical diagram of a NVDL schema.

Introduction

Oxygen XML Editor provides a simple, expressive, and easy to read Schema Diagram View for NVDL schemas.

With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, and BMP images. It helps both schema authors in developing the schema and content authors that are using the schema to understand it.

Oxygen XML Editor is the only XML Editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- the changes you make in the Editor are immediately visible in the Diagram (no background parsing).
- changing the selected element in the diagram, selects the underlying code in the source editor.

Full Model View

When you create a schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The diagram view has two tabbed panes offering a **Full Model View** and a **Logical Model View**.

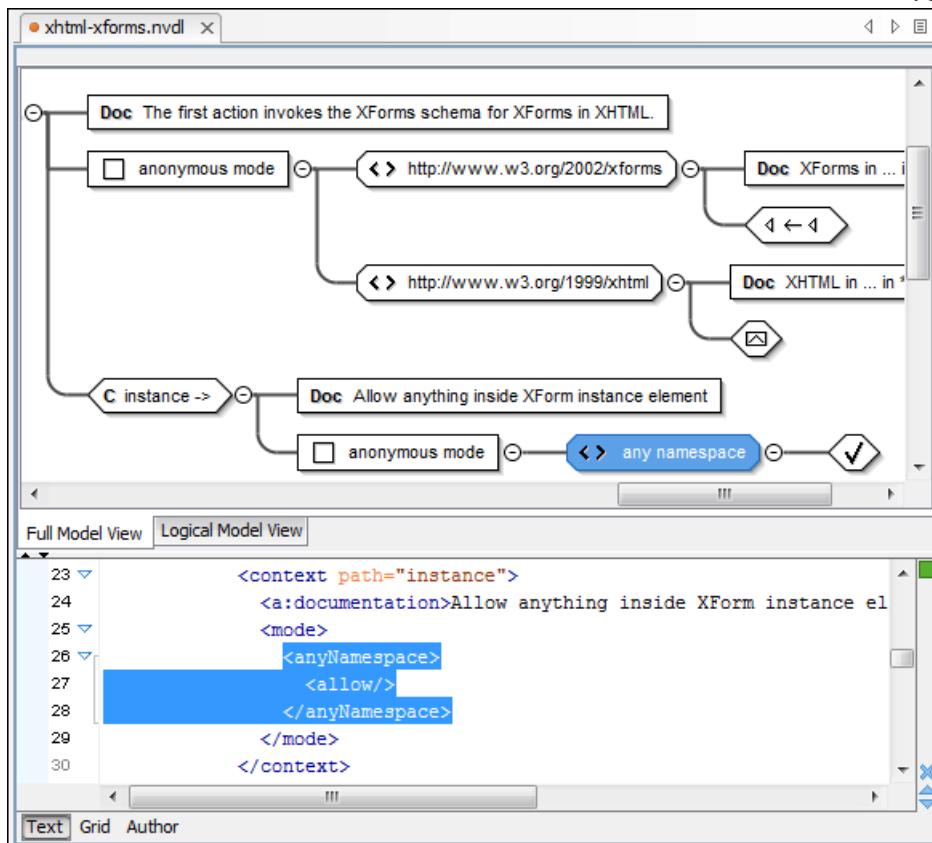


Figure 203: NVDL Schema Editor - Full Model View

The **Full Model View** renders all the NVDL elements with intuitive icons. This representation coupled with the synchronization support makes the schema navigation easy.

Double click on any diagram component in order to edit its properties.

Actions Available in the Diagram View

The contextual menu offers the following actions:

Show only the selected component

Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.

Show Annotations

Depending on its state (selected/not selected), the documentation nodes are shown or hidden.

Auto expand to references

This option controls how the schema diagram is automatically expanded. For instance, if you select it and then edit a top-level element or you trigger a diagram refresh, the diagram will be expanded until it reaches the referenced components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.

Collapse Children

Collapses the children of the selected view.

Expand Children

Expands the children of the selected view.

Print Selection...

Prints the selected view.

Save as Image...

Saves the current selection as image, in JPEG, BMP, SVG or PNG format.

Refresh

Refreshes the schema diagram according to the changes in your code (changes in your imported documents or those that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

NVDL Outline View

The NVDL Outline view presents a list with the named or anonymous rules that appear in the diagram. It allows a quick access to a rule by name. It can be opened from the **Window > Show View > Outline** menu.

NVDL Editor Specific Actions

The list of actions specific for the Oxygen XML Editor NVDL editor of is:

- **Document > Schema > Show Definition** (also available on the contextual menu of the editor panel) - Moves the cursor to its definition in the schema used by NVDL in order to validate it. You can use the **Ctrl Click (Command Click on OS X)** shortcut on a reference to display its definition.

Searching and Refactoring Actions in NVDL Schemas

Search Actions

The following search actions can be applied on `name`, `useMode`, and `startMode` attributes and are available from the **Search** submenu in the contextual menu of the current editor or from the **Document > References** menu:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on `name`, `useMode`, and `startMode` attributes and are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

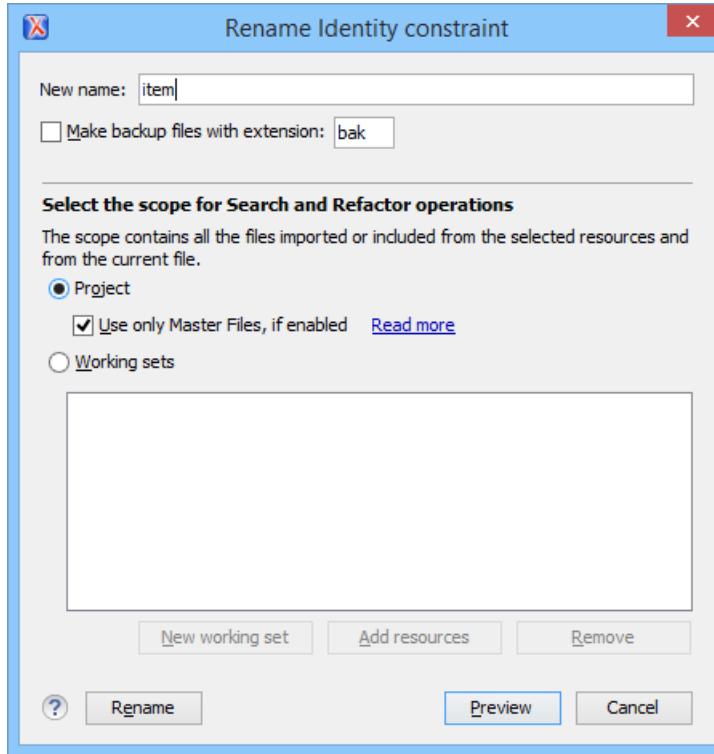


Figure 204: Rename Identity Constraint Dialog Box

Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected NVDL named mode. You can open the view from **Window > Show View > Component Dependencies**.

If you want to see the dependencies of an NVDL mode, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named modes.

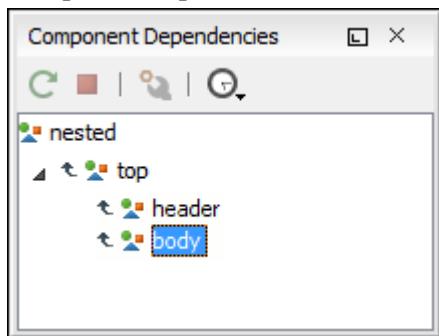


Figure 205: Component Dependencies View - Hierarchy for test.nvd1

In the **Component Dependencies** the following actions are available on the toolbar:



Refresh

Refreshes the dependencies structure.



Stop

Allows you to stop the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. If you decide to set the application to use automatically the defined scope for future operations, select the corresponding checkbox.

History

Repeats a previous dependencies computation.

The following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

 **Tip:** If a component contains multiple references to another component, a small table containing all references is shown. When a recursive reference is encountered it is marked with a special icon .

Linking Between Development and Authoring

The **Author** mode is available on the NVDL scripts editor presenting them in a compact and easy to understand representation.

Editing JSON Documents

This section explains the features of the Oxygen XML Editor JSON Editor and how to use them.

JSON Editor Text Mode

The **Text Mode** of the JSON editor provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, drag and drop, and other editor actions like *validation* and *formatting and indenting (pretty print) document*.

You can use the two **Text** and **Grid** buttons available at the bottom of the editor panel if you want to switch between the editor **Text Mode** and **Grid Mode**.

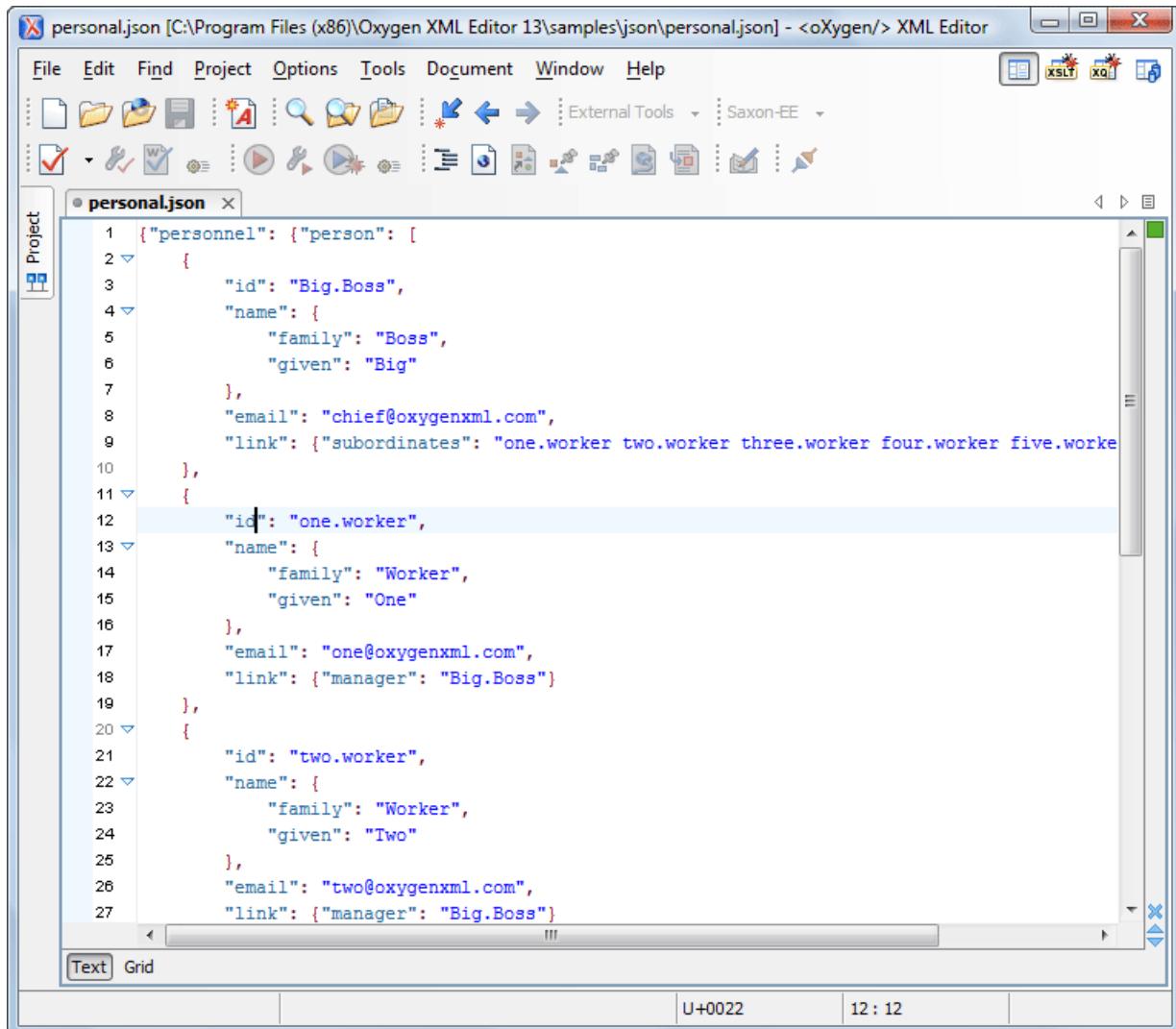


Figure 206: JSON Editor Text Mode

Syntax highlight in JSON Documents

Oxygen XML Editor supports *Syntax Highlight* for JavaScript / JSON editors and provides default configurations for the JSON set of tokens. You can customize the foreground color, background color and the font style for each JSON token type.

Folding in JSON

In a large JSON document, the data enclosed in the '{' and '}' characters can be collapsed so that only the needed data remain in focus. The [folding features available for XML documents](#) are available in JSON documents.

JSON Editor Grid Mode

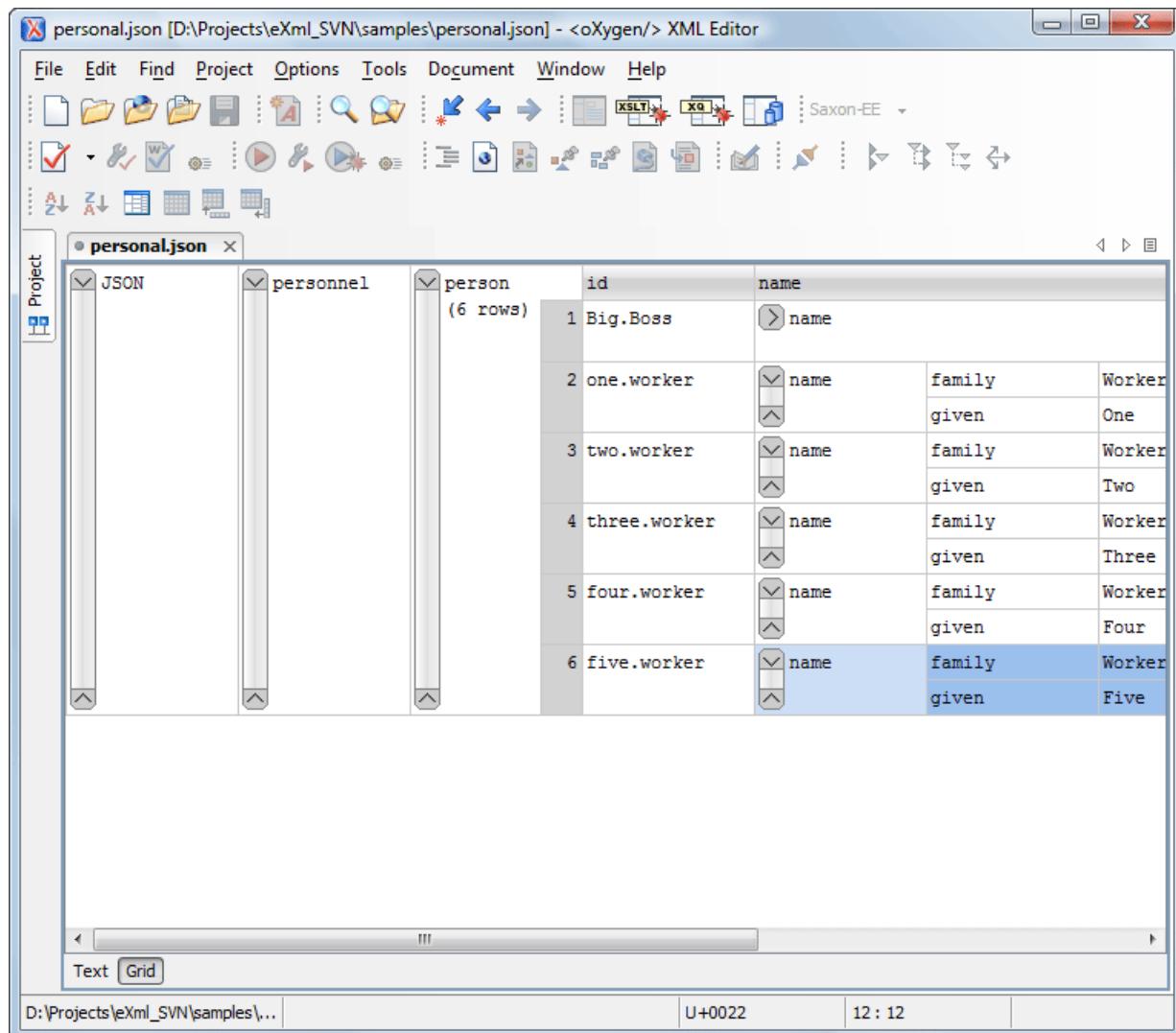


Figure 207: JSON Editor Grid Mode

Oxygen XML Editor allows you to view and edit the JSON documents in the *Grid Mode*. The JSON is represented in Grid mode as a compound layout of nested tables in which the JSON data and structure can be easily manipulated with table-specific operations or drag and drop operations on the grid components. You can also use the following JSON-specific contextual actions:

Array

Useful when you want to convert a JSON *value* to *array*.

Insert value before

Inserts a value before the currently selected one.

Insert value after

Inserts a value after the currently selected one.

Append value as child

Appends a value as a child of the currently selected value.

You can *customize the JSON grid appearance* according to your needs. For instance you can change the font, the cell background, foreground, or even the colors from the table header gradients. The default width of the columns can also be changed.

JSON Outline View

The **JSON Outline** view displays the list of all the components of the JSON document you are editing. To enable the **JSON Outline** view, go to **Window > Show view > Outline**.

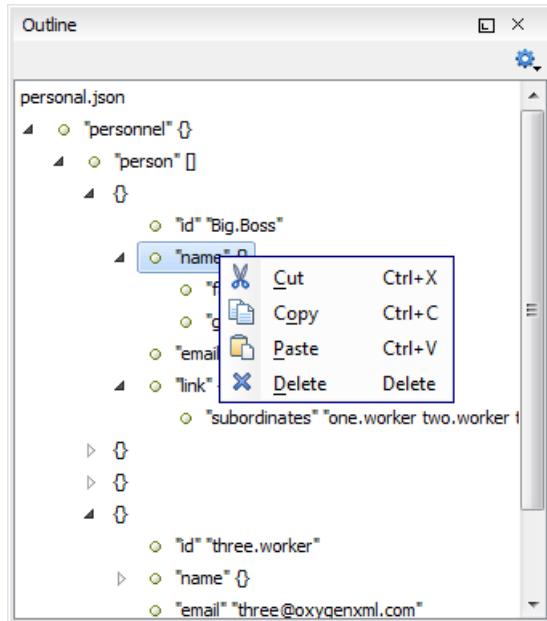


Figure 208: The JSON Outline View

The following actions are available in the contextual menu of the **JSON Outline** view:

- **Cut**
- **Copy**
- **Paste**
- **Delete**

The settings menu of the **JSON Outline** view allows you to enable **Selection update on caret move**. This option controls the synchronization between the **Outline** view and source the document. Oxygen XML Editor synchronizes the selection in the **Outline** view with the caret moves or the changes you make in the JSON editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Validating JSON Documents

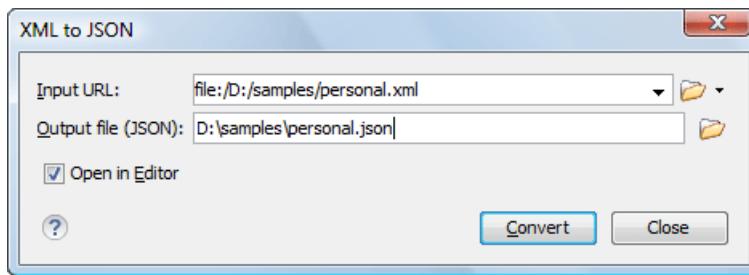
Oxygen XML Editor includes a built-in JSON validator (based on the free JAVA source code available on www.json.org), integrated with the general validation support.

Convert XML to JSON

The steps for converting an XML document to JSON are the following:

1. Go to menu **Tools > XML to JSON...**.

The **XML to JSON** dialog box is displayed:



2. Choose or enter the **Input URL** of the XML document.
3. Choose the **Output file** that will contain the conversion JSON result.
4. Check the **Open in Editor** option to open the JSON result of the conversion in the Oxygen XML Editor JSON Editor
5. Click the **OK** button.

The operation result will be a document containing the JSON conversion of the input XML URL.

```

personal.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE personnel SYSTEM "personnel.dtd">
3 <?xml-stylesheet type="text/css" href="personnel.css"?>
4 <personnel>
5   <person id="Big.Boss">
6     <name>
7       <family>Boss</family>
8       <given>Big</given>
9     </name>
10    <email>chief@oxygenxml.com</email>
11    <link subordinates="one.worker" manager="three.worker" />
12  </person>
13  <person id="one.worker">
14    <name>
15      <family>Worker</family>
16      <given>One</given>
17    </name>
18    <email>one@oxygenxml.com</email>
19    <link manager="Big.Boss" />
20  </person>
21  <person id="two.worker">
22    <name>
23      <family>Worker</family>
24      <given>Two</given>
25    </name>
26    <email>two@oxygenxml.com</email>
27    <link manager="Big.Boss" />
28  </person>
29  <person id="three.worker">
30    <name>
31      <family>Worker</family>
32      <given>Three</given>
33    </name>
34    <email>three@oxygenxml.com</email>
35    <link manager="one.worker" />
36  </person>
37 </personnel>
38 
```

```

personal.json
1 {"personnel": {"person": [
2   {
3     "id": "Big.Boss",
4     "name": {
5       "family": "Boss",
6       "given": "Big"
7     },
8     "email": "chief@oxygenxml.com",
9     "link": {"subordinates": "one.worker", "manager": "three.worker"}
10   },
11   {
12     "id": "one.worker",
13     "name": {
14       "family": "Worker",
15       "given": "One"
16     },
17     "email": "one@oxygenxml.com",
18     "link": {"manager": "Big.Boss", "subordinates": "two.worker"}
19   },
20   {
21     "id": "two.worker",
22     "name": {
23       "family": "Worker",
24       "given": "Two"
25     },
26     "email": "two@oxygenxml.com",
27     "link": {"manager": "Big.Boss", "subordinates": "three.worker"}
28   }
29 ]}}
30 
```

Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Editor supports StratML Part 1 (Strategic Plan) and StratML Part 2 (Performance Plans and Reports) and provides templates for the following documents:

- **Strategic Plan** (StratML Part 1)
- **Performance Plan** (StratML Part 2)
- **Performance Report** - (StratML Part 2)
- **Strategic Plan** - (StratML Part 2)

You can view the components of a StratML document in the **Outline** view. Oxygen XML Editor implements a default XML with XSLT transformation scenario for this document type, called StratML to HTML.

Editing JavaScript Documents

This section explains the features of the Oxygen XML Editor JavaScript Editor and how you can use them.

JavaScript Editor Text Mode

Oxygen XML Editor allows you to create and edit JavaScript files and assists you with useful features such as syntax highlight, content completion, and outline view. To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.

```

12 ▽function newPage(filename, overlay) {
13     divs = document.getElementsByTagName("div");
14
15     if (divs) {
16         var xdiv = divs[0];
17
18     if (xdiv) {
19         var xid = xdiv.getAttribute("id");
20
21         var mytoc = window.top.frames[0];
22         if (mytoc.lastUnderlined) {
23             mytoc.lastUnderlined.style.textDecoration = "none";
24         }
25
26         var tdiv = xbGetElementById(xid, mytoc);
27
28         if (tdiv) {
29             var ta = tdiv.getElementsByTagName("a").item(0);
30             ta.style.textDecoration = "underline";
31             mytoc.lastUnderlined = ta;
32         }
33     }
34 }
35
36     if (overlay != 0) {
37     overlaySetup('lc');
38 }
```

Figure 209: JavaScript Editor Text Mode

The contextual menu of the **JavaScript** editor offers the following actions:



Cut

Allows you to cut fragments of text from the editing area.



Copy

Allows you to copy fragments of text from the editing area.



Paste

Allows you to paste fragments of text in the editing area.



Toggle comment

Allows you to comment a line or a fragment of the JavaScript document you are editing. This option inserts a single comment for the entire fragment you want to comment.

 **Toggle line comment**

Allows you to comment a line or a fragment of the JavaScript document you are editing. This option inserts a comment for each line of the fragment you want to comment.

Go to matching bracket

Use this option to find the closing, or opening bracket, matching the bracket at the caret position. When you select this option, Oxygen XML Editor moves the caret to the matching bracket, highlights its row, and decorates the initial bracket with a rectangle.



Note: A rectangle decorates the opening, or closing bracket which matches the current one at all times.

Compare

Select this option to open the **Diff Files** dialog and compare the file you are editing with a file you choose in the dialog.

Open

Allows you to select one of the following actions:

- **Open File at Caret** - select this action to open the source of the file located at the caret position
- **Open File at Caret in System Application** - select this action to open the source of the file located at the caret position with the application that the system associates with the file

Folding

Allows you to select one of the following actions:

 **Toggle Fold**

Toggles the state of the current fold.

 **Collapse Other Folds (Ctrl (Meta on Mac OS)+NumPad /)**

Folds all the elements except the current element.

 **Collapse Child Folds (Ctrl (Meta on Mac OS)+NumPad .)**

Folds the elements indented with one level inside the current element.

 **Expand Child Folds**

Unfolds all child elements of the currently selected element.

 **Expand All (Ctrl (Meta on Mac OS)+NumPad *)**

Unfolds all elements in the current document.

Source

Allows you to select one of the following actions:

To Lower Case

Converts the selection content to lower case characters.

To Upper Case

Converts the selection content to upper case characters.

Capitalize Lines

Converts to upper case the first character of every selected line.

Join and Normalize Lines

Joins all the rows you select to one row and normalizes the content.

Insert new line after

Inserts a new line after the line at the caret position.

Content Completion in JavaScript Files

When you edit a JavaScript document, the **Content Completion Assistant** presents you a list of the elements you can insert at the caret position. For an enhanced assistance, JQuery methods are also presented. The following icons decorate the elements in the content completion list of proposals depending on their type:

- - function
- - variable
- - object
- - property
- - method

Note: These icons decorate both the elements from the content completion list of proposals and from the **Outline** view.

```

12  function newPage(filename, overlay) {
13      divs = document.getElementsByTagName("div");
14
15  if (divs) {
16      var xdiv = divs[0];
17
18  if (xdiv) {
19      var xid = TypeInfo - TypeInfo
20      UIEvent - UIEvent
21      UserDataHandler - UserDataHandler
22  if (mytoc) alert(msg)
23      mytoc.lastUnderlined = ta;
24      blur()
25      clearInterval(id_setinterval)
26      clearTimeout(id_settimeout)
27      var tdiv = document.createElement("div");
28
29  if (tdiv) {
30      var ta = tdiv.getElementsByTagName("a").item(0);
31      ta.style.textDecoration = "underline";
32      mytoc.lastUnderlined = ta;
33  }
34  }
35
36  if (overlay != 0) {
37      overlaySetup('lc');
38  }

```

Figure 210: JavaScript Content Completion Assistant

The **Content Completion Assistant** collects:

- Method names from the current file and from the library files.
- Functions and variables defined in the current file.

In case you edit the content of a function, the content completion list of proposals contains all the local variables defined in the current function, or in the functions that contain the current one.

JavaScript Outline View

Oxygen XML Editor present a list of all the components of the JavaScript document you are editing in the **Outline** view. To open the **Outline** view, go to **Window > Show View > Outline**.

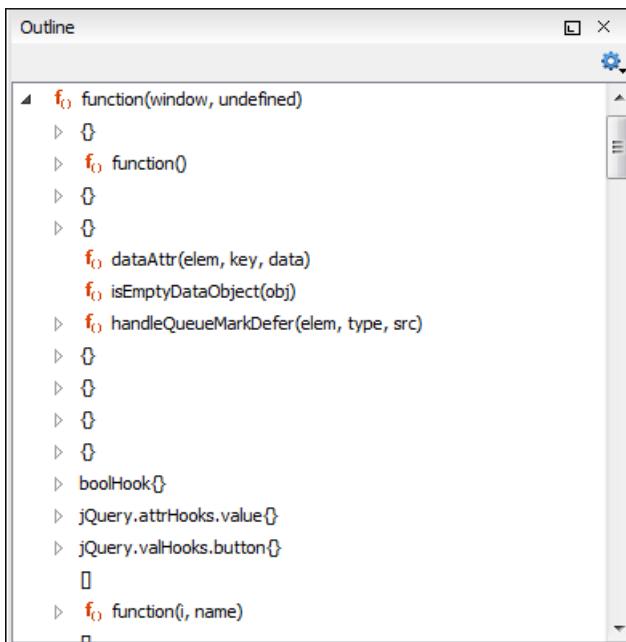


Figure 211: The JavaScript Outline View

The following icons decorate the elements in the **Outline** view depending on their type:

- - function
- - variable
- - object
- - property
- - method

The contextual menu of the JavaScript **Outline** view contains the usual Cut, Copy, Paste, and Delete actions. From the settings menu, you can enable the Update selection on caret move option to synchronize the **Outline** view with the editing area.

Validating JavaScript Files

You have the possibility to validate the JavaScript document you are editing. Oxygen XML Editor uses the Mozilla Rhino library for validation. For more information about this library, go to <http://www.mozilla.org/rhino/doc.html>. The JavaScript validation process checks for errors in the syntax. Calling a function that is not defined is not treated as an error by the validation process. The interpreter discovers this error when executing the faulted line. Oxygen XML Editor can validate a JavaScript document both on-request and automatically.

Editing XProc Scripts

An XProc script is edited as an XML document that is validated against a RELAX NG schema. If the script has an associated transformation scenario, then the XProc engine from the scenario is invoked as validating engine. The default engine for XProc scenarios is the Calabash engine which comes bundled with Oxygen XML Editor version 17.0.

The content completion inside the element `input` / `inline` from the XProc namespace <http://www.w3.org/ns/xproc> offers elements from the following schemas depending both on the `port` attribute and the parent of the `input` element. When invoking the content completion inside the XProc element `inline`, the **Content Completion Assistant**'s proposals list is populated as follows:

- If the value of the `port` attribute is `stylesheet` and the `xslt` element is the parent of the `input` elements, the **Content Completion Assistant** offers XSLT elements.

- If the value of the port attribute is schema and the validate-with-relax-ng element is the parent of the input element, the **Content Completion Assistant** offers RELAX NG schema elements.
- If the value of the port attribute is schema and the validate-with-xml-schema element is the parent of the input element, the **Content Completion Assistant** offers XML Schema schema elements.
- If the value of the port attribute is schema and the validate-with-schematron element is the parent of the input element, the **Content Completion Assistant** offers either ISO Schematron elements or Schematron 1.5 schema elements.
- If the above cases do not apply, then the **Content Completion Assistant** offers elements from all the schemas from the above cases.

The XProc editor assists you in writing XPath expressions by offering a **Content Completion Assistant** and dedicated coloring schemes. To customize the coloring schemes, [open the Preferences dialog box](#) and go to **Colors**.

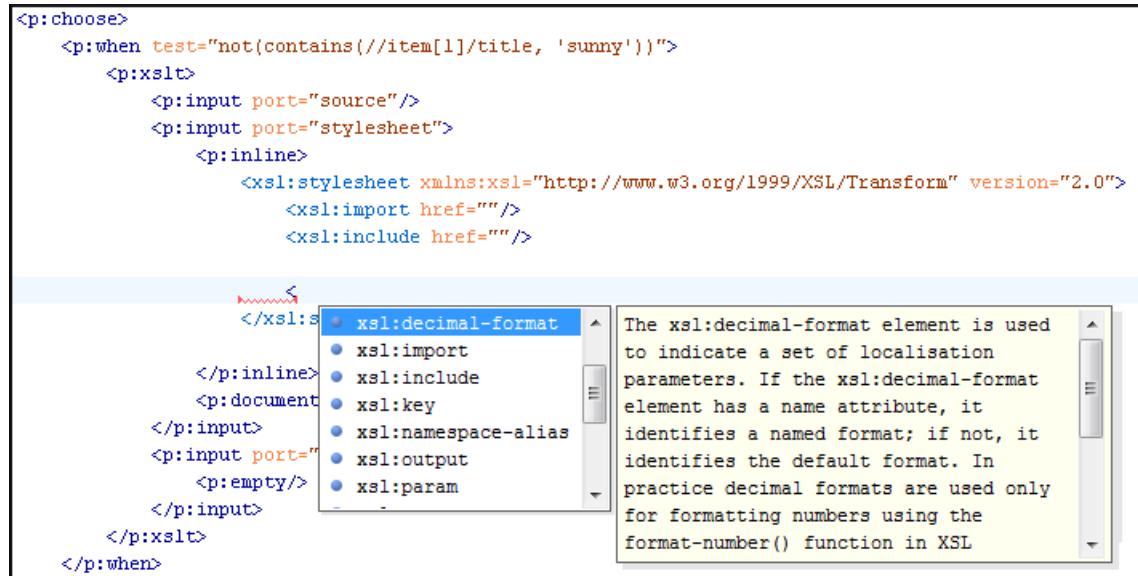


Figure 212: XProc Content Completion

Editing Schematron Schemas

Schematron is a simple and powerful Structural Schema Language for making assertions about patterns found in XML documents. It relies almost entirely on XPath query patterns for defining rules and checks. Schematron validation rules allow you to specify a meaningful error message. This error message is provided to you if an error is encountered during the validation stage.

The Skeleton XSLT processor is used for validation and conforms with ISO Schematron or Schematron 1.5. It allows you to validate XML documents against Schematron schemas or against combined RELAX NG / W3C XML Schema and Schematron.

Oxygen XML Editor assists you in editing Schematron documents with schema-based content completion, syntax highlight, search and refactor actions, and dedicated icons for the **Outline** view. You can create a new Schematron schema using one of the Schematron templates available in the New Document wizard.

The Schematron editor renders with dedicated coloring schemes the XPath expressions. To customize the coloring schemes, [open the Preferences dialog box](#) and go to **Editor > Colors**.

To watch our video demonstration about the Schematron support in Oxygen XML Editor, go to

http://www.oxygenxml.com/demo/Schematron_Validation.html, and to

http://www.oxygenxml.com/demo/Editing_Schematron_Schemas/Editing_Schematron_Schemas.html.



Note: When you create a Schematron document, Oxygen XML Editor provides a built-in transformation scenario. You are able to use this scenario to obtain the XSLT style-sheet corresponding to the Schematron schema. You can apply this XSLT stylesheet to XML documents to obtain the Schematron validation results.

Validate an XML Document

To validate an XML document against a Schematron schema, select the **Validate** action from the **Validation** toolbar drop-down list, the **Document > Validate** menu, or from the **Validate** menu when invoking the contextual menu in the **Project** view. If you would like to add a persistence association between your Schematron rules and the current edited XML document, use the **Associate Schema** action from the **Document > Schema** menu or the **Document** toolbar. A custom processing instruction is added into the document and the validation process will take into account the Schematron rules:

```
<?xml-model href="percent.sch" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

The possible errors which might occur during the validation process are presented in the **Errors** panel at the bottom area of the Oxygen XML Editor window. Each error is flagged with a severity level that can be one of *warning*, *error*, *fatal* or *info*.

To set a severity level, Oxygen XML Editor looks for the following information:

- The `role` attribute, which can have one of the following values:
 - `warn` or `warning`, to set the severity level to *warning*
 - `error`, to set the severity level to *error*
 - `fatal`, to set the severity level to *fatal*
 - `info` or `information`, to set the severity level to *info*
 - The start of the message, after trimming leading white-spaces. Oxygen XML Editor looks to match the following exact string of characters (case sensitive):
 - `Warning:`, to set the severity level to *warning*
 - `Error:`, to set the severity level to *error*
 - `Fatal:`, to set the severity level to *fatal*
 - `Info:`, to set the severity level to *info*
- Note:** Displayed message does not contain the matched prefix.
- If none of the previous rules match, Oxygen XML Editor sets the severity level to *error*.

Validating Schematron Documents

By default, a Schematron schema is validated as you type. To change this, [open the Preferences dialog box](#), go to **Editor > Document Checking**, and disable the **Enable automatic validation** option.

To validate your Schematron document manually, select the **Validate** action from the **Validation** toolbar drop-down list or the **Document > Validate** menu. When Oxygen XML Editor validates a Schematron schema, it expands all the included modules so the entire schema hierarchy is validated. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

Oxygen XML Editor offers an error management mechanism capable of pinpointing errors in XPath expressions and in the included schema modules.

Content Completion in Schematron Documents

Oxygen XML Editor helps you edit a Schematron schema, offering, through the Content Completion Assistant, items that are valid at the caret position. When you edit the value of an attribute that refers a component, the proposed

components are collected from the entire schema hierarchy. For example, if the editing context is `phase/active/@pattern`, the Content Completion Assistant proposes all the defined patterns.



Note: For Schematron resources, the Content Completion Assistant collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to [Defining Master Files at Project Level](#).

If the editing context is an attribute value that is an XPath expression (like `assert/@test` or `report/@test`), the Content Completion Assistant offers the names of XPath functions, the XPath axes, and user-defined variables.

The **Content Completion Assistant** displays XSLT 1.0 functions and optionally XSLT 2.0 / 3.0 functions in the attributes `path`, `select`, `context`, `subject`, `test` depending on [the Schematron options](#) that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.6.0.5 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion also displays the XSLT Saxon extension functions as in the following figure:

The screenshot shows the Oxygen XML Editor interface with code completion enabled. A tooltip is displayed over the word `id` in the code, providing a detailed explanation of the `saxon:id` function. The code itself is a Schematron schema snippet, likely a pattern for teams, with various rules and assertions.

```

48 <sch:rule context="t:Type[t = 'Doubles']">
49   <sch:assert test=".../t:Par[generate-id(node-set?)>
50     you're playing doubles t[id(object)
51       divisible by 2.</sch:assert>
52     <sch:assert test=".../t:Par[lang(string)
53       .../t:Teams/@nbrTeams * 2]>
54       of participants must equ[local-name(node-set?)]
55         f[ name(node-set?)]>
56   <sch:rule context="t:Participants">
57     <sch:assert test="count(t:id(object) as nodeset
58       Name elements in <sch:na
59         attribute.</sch:assert>>
60       The id function selects elements by
61       their unique ID. When the argument to
62       id is of type node-set, then the result
63       is the union of the result of applying
64       id to the string-value of each of the
65       nodes in the argument node-set. When
66       the argument to id is of any other
67     <sch:pattern id="Teams">
68       <sch:rule context="t:Teams">
69         <sch:assert diagnostics="d1" test="count(t:Team) = @nbrTeams">The

```

Figure 213: XSLT extension functions in Schematron schemas documents

The **Content Completion Assistant** also includes [code templates that can be used to quickly insert code fragments](#) into Schematron documents.

RELAX NG/XML Schema with Embedded Schematron Rules

Schematron rules can be embedded into an XML Schema through annotations (using the `appinfo` element), or in any element on any level of a RELAX NG Schema (taking into account that the RELAX NG validator ignores all elements that are not in the RELAX NG namespace).

Oxygen XML Editor accepts such documents as Schematron validation schemas and it is able to extract and use the embedded rules. To validate an XML document with both RELAX NG schema and its embedded Schematron rules, you need to associate the document with both schemas. For example:

```

<?xml-model href="percent.rng" type="application/xml"
schematypens="http://relaxng.org/ns/structure/1.0"?>
<?xml-model href="percent.rng" type="application/xml"
schematypens="http://purl.oclc.org/dsdl/schematron"?>

```

The second association validates your document with Schematron rules extracted from the RELAX NG Schema.

Similarly, you can specify an XML Schema having the embedded Schematron rules.

```
<?xml-model href="percent.xsd" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>
```



Note: When you work with XML Schema or Relax NG documents that have embedded Schematron rules Oxygen XML Editor provides two built-in validation scenarios: **Validate XML Schema with embedded Schematron** for XML schema , and **Validate Relax NG with embedded Schematron** for Relax NG. You can use one of these scenarios to validate the embedded Schematron rules.

Editing Schematron Schema in the Master Files Context

Smaller interrelated modules that define a complex Schematron cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a diagnostic defined in a main schema document is not visible when you edit an included module. Oxygen XML Editor provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You can set a main Schematron document either using the *master files support from the Project view*, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Editor warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- Correct validation of a module in the context of a larger schema structure.
- **Content Completion Assistant** displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one.

Schematron Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a Schematron schema. To open this view, go to **Window > Show View > Resource Hierarchy/Dependencies**.

If you want to see the hierarchy of a schema, select the desired schema in the project view and choose **Resource Hierarchy** from the contextual menu.

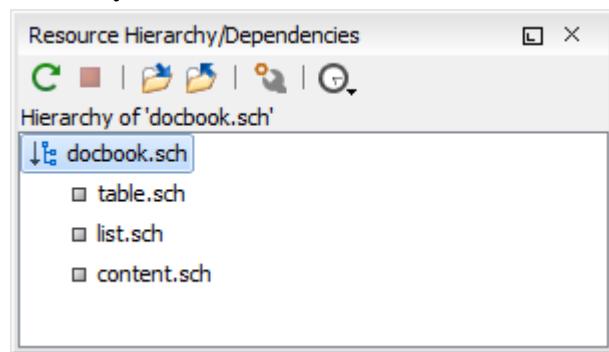


Figure 214: Resource Hierarchy/Dependencies View

If you want to see the dependencies of a schema, select the desired schema in the project view and choose **Resource Dependencies** from the contextual menu.

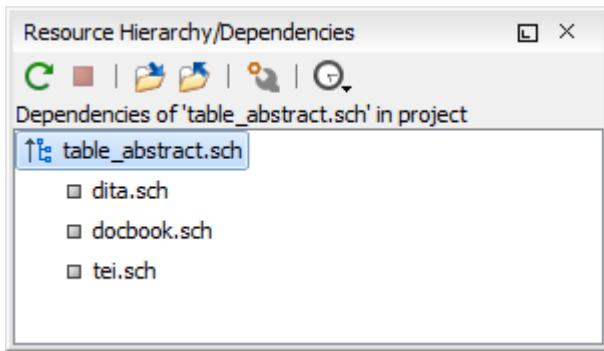


Figure 215: Resource Hierarchy/Dependencies View - Dependencies for `table_abstract.sch`

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

-  **Tip:** When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .

-  **Note:** The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming Schematron Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Highlight Component Occurrences in Schematron Documents

When you position your mouse cursor over a component in a Schematron document, Oxygen XML Editor searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default behaviour of **Highlight Component Occurrences**, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. You can also trigger a search using the **Search > Search Occurrences in File Ctrl Shift U (Command Shift U on OS X)** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in Schematron Documents

Search Actions

The following search actions can be applied on pattern, phase, or diagnostic types and are available from the **Search** submenu in the contextual menu of the current editor or from the **Document > References** menu:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on pattern, phase, or diagnostic types and are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

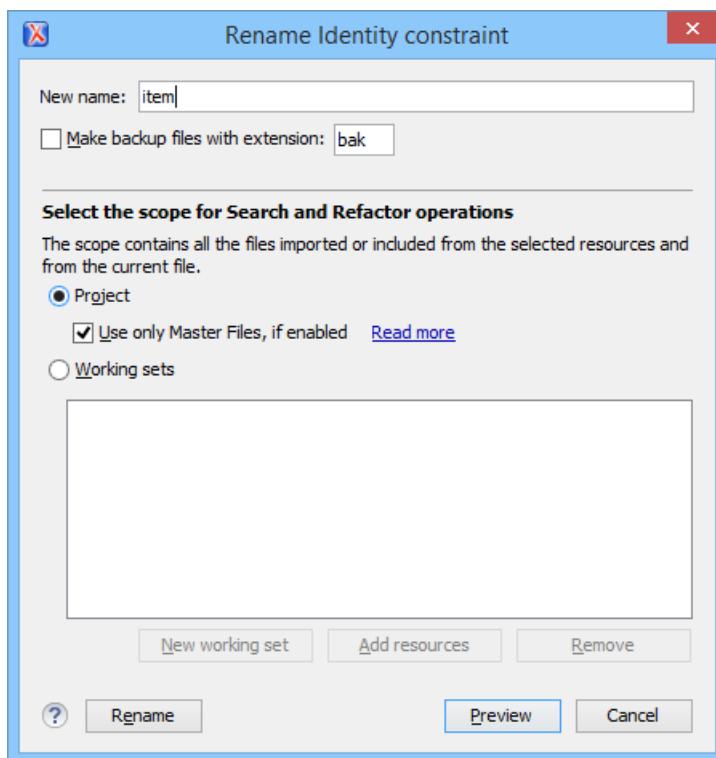


Figure 216: Rename Identity Constraint Dialog Box

Searching and Refactoring Operations Scope in Schematron Documents

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the  **Change scope** operation, available in the Quick Assist action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.

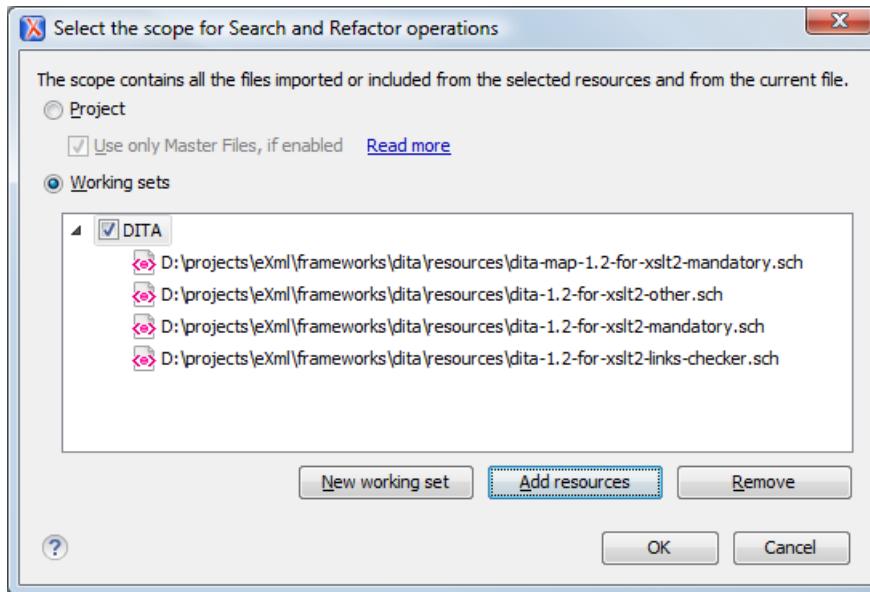


Figure 217: Change Scope Dialog

The scope you define is applied to all future search and refactor operations until you modify it. Contextual menu actions allow you to add or delete files, folders, and other resources to the working set structure.

Quick Assist Support in Schematron Documents

Quick Assist is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X) on your keyboard.

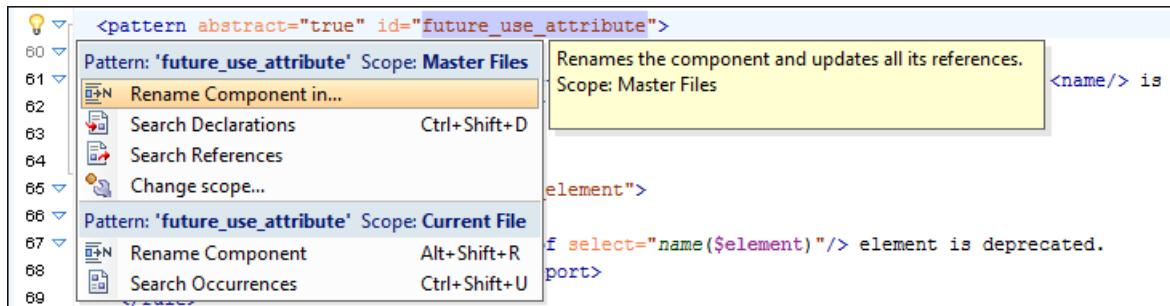


Figure 218: Schematron Quick Assist Support

The quick assist support offers direct access to the following actions:

Rename Component in...

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope...

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Editing Schematron Quick Fixes

Oxygen XML Editor provides support for editing the *Schematron Quick Fixes*. You can define a library of quick fixes by editing them directly in the current Schematron file or in a separate file. Oxygen XML Editor assists you in editing Schematron Quick Fixes with schema-based content completion, syntax highlighting, and validation as you type.

This section includes details about the Schematron Quick Fixes feature and how to customize them.

Validating Schematron Quick Fixes

By default, Schematron Quick Fixes are validated as you edit them within the Schematron file or while editing them in a separate file. To change this, [open the Preferences dialog box](#), go to **Editor > Document Checking**, and disable the **Enable automatic validation** option.

To validate Schematron Quick Fixes manually, select the  **Validate** action from the  **Validation** toolbar drop-down list or the **Document > Validate** menu. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

Content Completion in SQF

Oxygen XML Editor helps you edit Schematron Quick Fixes embedded in a Schematron document, offering, through the Content Completion Assistant, items that are valid at the caret position. When you edit the value of an attribute that refers a quick fix *id*, the ids are collected from the entire definition scope. For example, if the editing context is `assert/@sqf:fix`, the Content Completion Assistant proposes all fixes defined locally and globally.

If the editing context is an attribute value that is an XPath expression (such as `sqf:add/@match` or `replace/@select`), the Content Completion Assistant offers the names of XPath functions, the XPath axes, and user-defined variables and parameters.

The **Content Completion Assistant** displays XSLT 1.0 functions (and optionally XSLT 2.0 / 3.0 functions) in the attributes *path*, *select*, *context*, *subject*, and *test*, depending on [the Schematron options](#) that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.6.0.5 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion also displays the XSLT Saxon extension functions.

Highlight Quick Fix Occurrences in SQF

When you position your mouse cursor over a quick fix id in a Schematron document, Oxygen XML Editor searches for the quick fix declaration and all its references and highlights them automatically.

Customizable colors are used: one for the quick fix definition and another one for its references. Occurrences are displayed until another quick fix is selected.

To change the default behaviour of **Highlight Component Occurrences**, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. You can also trigger a search using the **Search > Search Occurrences in File (Ctrl Shift U (Command Shift U on OS X))** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in SQF

Search Actions

The following search actions can be applied on quick fix ids and are available from the **Search** submenu in the contextual menu of the current editor or from the **Document > References** menu:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on quick fix ids and are available from the **Refactoring** submenu in the contextual menu of the current editor or from the **Document > Refactoring** menu:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

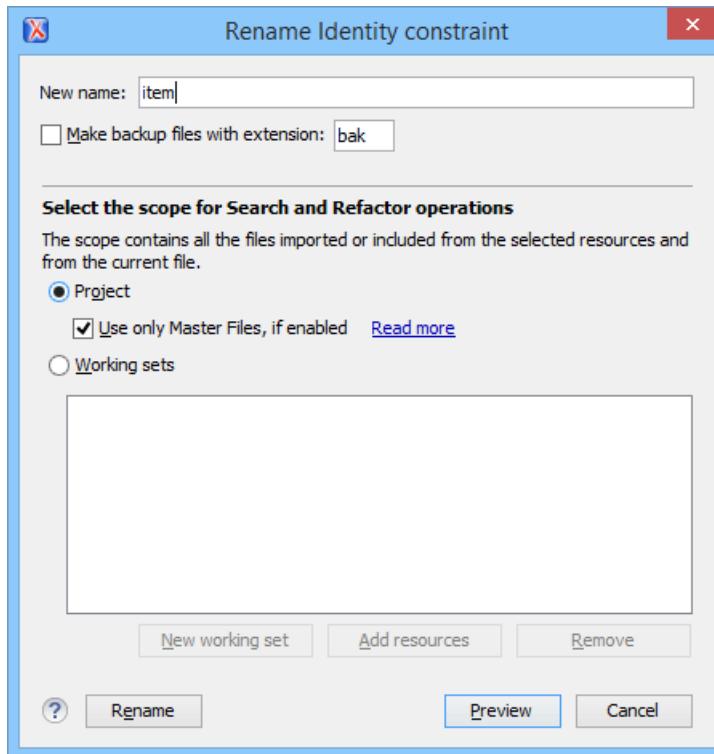


Figure 219: Rename Identity Constraint Dialog Box

Embed Schematron Quick Fixes in Relax NG or XML Schema

Schematron Quick Fixes can be embedded into a Relax NG or XML Schema within the Schematron rules from annotations (using the `appinfo` element), or in any Schematron rule of a RELAX NG Schema.

Oxygen XML Editor is able to extract and use the embedded Schematron Quick Fixes. To make the Schematron Quick Fixes available, validate the document with both the RELAX NG schema and its embedded Schematron rules.

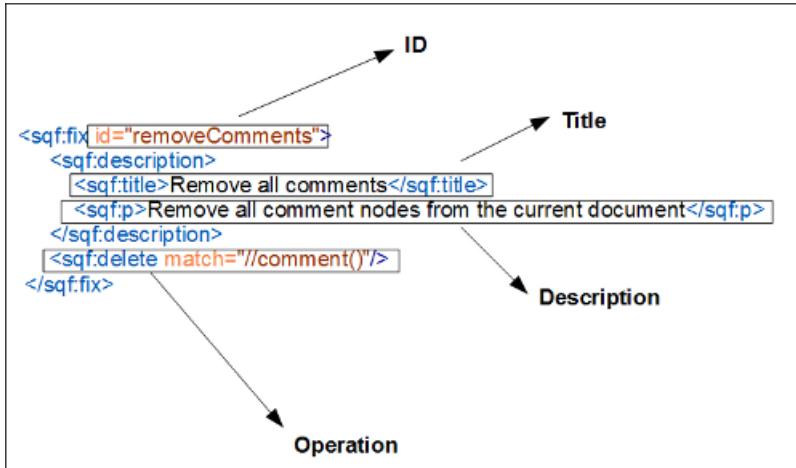
Customizing Schematron Quick Fixes

You can customize Schematron Quick Fixes by editing them directly in the current Schematron file or in a separate file. The Schematron Quick Fixes are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. You can refer the quick fixes from the `assert` or `report` elements in the values of the `sqf:fix` attributes.

Defining a Schematron Quick Fix

The basics of a Schematron Quick Fix is defined by an ID, name, description, and the operations to be executed.

- **ID** - Defined by the `id` attribute from the `fix` element and must be unique in the current context. It is used to refer the quick fix from a `report` or `assert` element.
- **Name** - The name of the quick fix is defined by the `title` element.
- **Description** - Defined by the text in the paragraphs (`p`) of the `description` element.
- **Operations** - The following types of operations are supported:
 - `<sqf:add>` - To add a new node or fragment in the document.
 - `<sqf:delete>` - To remove a node from the document.
 - `<sqf:replace>` - To replace a node with another node or fragment.
 - `<sqf:stringReplace>` - To replace text content with other text or a fragment.



The assertion message that generates the quick fix is added as the `description` of the problem to be fixed. The `title` is presented as the name of the quick fix. The content of the paragraphs (`p`) within the `description` element are presented in the tooltip message when the quick fix is selected.

Schematron Quick Fix Operations

Add

The `<sqf : add>` element allows you to add a node to the instance. An anchor node is required to select the position for the new node. The anchor node can be selected by the `match` attribute. Otherwise, it is selected by the `context` attribute of the rule.

The `target` attribute defines the name of the node to be added. It is required if the value of the `node-type` attribute is set to anything other than "comment".

The `<sqf : add>` element has a `position` attribute and it determines the position relative to the anchor node. The new node could be specified as the first child of the anchor node, the last child of the anchor node, before the anchor node, or after the anchor node (`first-child` is the default value). If you want to add an attribute to the anchor node, do not use the `position` attribute.

 **Note:** If you insert an element and its content is empty, Oxygen XML Editor will insert the required element content.

An Example of the `<sqf : add>` Element:

```

<schema xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
  <pattern>
    <rule context="head">
      <assert test="title" sqf:fix="addTitle">title element is missing.</assert>
      <sqf:fix id="addTitle">
        <sqf:description>
          <sqf:title>Insert title element.</sqf:title>
        </sqf:description>
        <sqf:add target="title" node-type="element">Title text</sqf:add>
      </sqf:fix>
    </rule>
  </pattern>
</schema>

```

Specific Add Operations:

- **Insert Element** - To insert an element, use the `<sqf : add>` element, set the value of the `node-type` to "element", and specify the element *QName* with the `target` attribute. If the element has a prefix, it must be defined in the Schematron using a namespace declaration (`<ns uri="namespace" prefix="prefix"/>`).
- **Insert Attribute** - To insert an attribute, use the `<sqf : add>` element, set the value of the `node-type` to "attribute", and specify the attribute *QName* with the `target` attribute. If the attribute has a prefix, it must be defined in the Schematron using a namespace declaration (`<ns uri="namespace" prefix="prefix"/>`).

- **Insert Fragment** - If the node-type is not specified, the `<sqf: add>` element will insert an XML fragment. The XML fragment must be well formed. You can specify the fragment in the add element or by using the select attribute.
- **Insert Comment** - To insert a comment, use the `<sqf: add>` element and set the value of the node-type to "comment".
- **Insert Processing Instruction** - To insert a processing instruction, use the `<sqf: add>` element, set the value of the node-type to "pi" or "processing-instruction", and specify the name of the processing instruction in the target attribute.

Delete

The `<sqf: delete>` element allows you to remove any type of node (such as elements, attributes, text, comments, or processing instructions). To specify nodes for deletion the `<sqf: delete>` element can include a match attribute that is an XPath expression (the default value is `.`). If the match attribute is not included, it deletes the context node of the Schematron rule.

An Example of the `<sqf: delete>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2">
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process">
  <pattern>
    <rule context="*[@xml:lang]">
      <report test="@xml:lang" sqf:fix="remove_lang">
        The attribute "xml:lang" is forbidden.</report>
      <sqf:fix id="remove_lang">
        <sqf:description>
          <sqf:title>Remove "xml:lang" attribute</sqf:title>
        </sqf:description>
        <sqf:delete match="@xml:lang"/>
      </sqf:fix>
    </rule>
  </pattern>
</schema>
```

Replace

The `<sqf: replace>` element allows you to replace nodes. Similar to the `<sqf: delete>` element, it can include a match attribute. Otherwise, it replaces the context node of the rule. The `<sqf: replace>` element has three tasks. It identifies the nodes to be replaced, defines the replacing nodes, and defines their content.

An Example of the `<sqf: replace>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
  <pattern>
    <rule context="title">
      <report test="exists(ph)" sqf:fix="resolvePh" role="warn">
        ph element is not allowed in title.</report>
      <sqf:fix id="resolvePh">
        <sqf:description>
          <sqf:title>Change the ph element into text</sqf:title>
        </sqf:description>
        <sqf:replace match="ph">
          <value-of select=".">

```

Other Attributes for Replace Operations:

- **node-type** - Determines the type of the replacing node. The permitted values include:
 - **keep** - Keeps the node type of the node to be replaced.
 - **element** - Replaces the node with an element.
 - **attribute** - Replaces the node with an attribute.
 - **pi** - Replaces the node with a processing instruction.
 - **comment** - Replaces the node with a comment.
- **target** - By using a *QName* it gives the replacing node a name. This is necessary when the value of the node-type attribute is anything other than "comment".

- **select** - Allows you to choose the content of the replacing nodes. You can use XPath expressions with the `select` attribute. If the `select` attribute is not specified then the content of the `<sqf:replace>` element is used instead.

String Replace

The `<sqf:stringReplace>` element is different from the others. It can be used to find a sub-string of text content and replace it with nodes or other strings.

Attributes for the String Replace Operation:

- **match** - Allows you to select text nodes that contain the sub-strings you want to replace.
- **select** - Allows you to select the replacing fragment, in case you do not want to set it in the content of the `stringReplace` element.
- **regex** - Matches the sub-strings using a regular expression.



Note: Regular expressions in the `<sqf:stringReplace>` element always has the *dot matches all* flag set to "true". Therefore, the line terminator will also be matched by the regular expression.



Attention: The context of the content within the `<sqf:stringReplace>` element is set to the whole text node, rather than the current sub-string.

An Example of the `<sqf:stringReplace>` Element:

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  xmlns:sql="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
  <sch:pattern>
    <sch:rule context="text()">
      <sch:report test="matches(., '[oO][xX]ygen')" sqf:fix="changeWord">The oXygen word is not
allowed</sch:report>
      <sqf:fix id="changeWord">
        <sqf:description>
          <sqf:title>Replace word with product</sqf:title>
        </sqf:description>
        <sqf:stringReplace regex="[oO][xX]ygen"><ph keyref="product"/></sqf:stringReplace>
      </sqf:fix>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Formatting and Indenting Inserted Content

The content that is inserted by the **Add**, **Replace**, or **String Replace** operations is automatically indented unless you set the value of the `xml:space` attribute to `preserve` on the operation element. There are several methods available to format the content that is inserted:

- **xsl:text** - You can use an `xsl:text` element to format the inserted content and keep the automatic indentation, as in the following example:

```
<sqf:add position="last-child">
  <row><xsl:text>
    </xsl:text>
    <entry>First column</entry><xsl:text>
    </xsl:text>
    <entry>Second column</entry><xsl:text>
    </xsl:text>
  </row><xsl:text>
  </xsl:text>
</sqf:add>
```

- **xml:space** - Use the `xml:space` attribute and set its value to `preserve` to format the content and specify the spacing between elements, as in the following example:

```
<sqf:add node-type="element" target="codeblock" xml:space="preserve">
  /* a long sample program */
  Do forever
    Say "Hello, World"
  End</sqf:add>
```

The `use-when` Condition

To restrict a quick fix or a specific operation to only be available if certain conditions are met, the `use-when` attribute can be included in the `<sqf:fix>` element or any of the SQF operation elements. The condition of the `use-when` attribute is an XPath expression and the fix or operation will be performed only if the condition is satisfied. In the following example, the `use-when` condition is applied to the `<sqf:fix>` element:

```
<sqf:fix id="last" use-when="$colWidthSummarized - 100 lt $lastWidth" role="replace">
    <sqf:description>
        <sqf:title>Subtract the excessive width from the last element.</sqf:title>
    </sqf:description>
    <let name="delta" value="$colWidthSummarized - 100"/>
    <sqf:add match="html:col[last()]" target="width" node-type="attribute">
        <let name="newWidth" value="number(substring-before(@width,'%')) - $delta"/>
        <value-of select="concat($newWidth,'%')"/>
    </sqf:add>
</sqf:fix>
```

Additional Elements Supported in the Schematron Quick Fixes

`<sqf:call-fix>`

This element calls another quick fix within a quick fix. The called quick fix must be defined globally or in the same Schematron rule as the calling quick fix. A calling quick fix adopts the activity elements of the called quick fix and should not include other activity elements. You can also specify which parameters are sent by using the `<sqf:with-param>` child element.

`<sqf:group>`

Allows you to group multiple quick fixes and refer them from an `assert` or `report` element.

`<sqf:fixes>`

Is defined globally and contains global fixes and groups of fixes.

`<sqf:keep>`

Used to copy the selected nodes that are specified by the `select` attribute.

 **Note:** In Oxygen XML Editor the copied nodes cannot be manipulated by the current or other activity elements.

`<sqf:param>`

Defines a parameter for a quick fix. If the parameter is defined as `abstract` then the `type` and `default` value should not be specified and the fix can be called from an abstract pattern that defines this parameter.

 **Warning:** The `<sqf:user-entry>` element is not supported and if it is used then the quick fix will not be displayed.

 **Note:** The `sqf:defaultFix` attribute is also ignored in Oxygen XML Editor.

For more details on editing Schematron Quick Fixes, go to: <http://www.schematron-quickfix.com/quickFix/reference.html>.

Editing SVG Documents

SVG is a platform for two-dimensional graphics. It has two parts: an XML-based file format and a programming API for graphical applications. Just to enumerate some of the key features: shapes, text, and embedded raster graphics with many painting styles, scripting through languages such as ECMAScript and support for animation.

SVG is a vendor-neutral open standard that has important industry support. Companies like Adobe, Apple, IBM, and others have contributed to the W3C specification. Many documentation frameworks, including DocBook, have support for SVG by defining the graphics directly in the document.

Oxygen XML Editor adds SVG support by using the *Batik* package, an open source project developed by the Apache Software Foundation. *Oxygen XML Editor's default XML catalog* solves the SVG DTD.



Note: Batik partially supports SVG 1.1. Here you can find a detailed list of supported elements, attributes and properties: [Batik implementation status](#).

To render SVG images which use Java scripting, copy the `js.jar` library from the Batik distribution into the Oxygen XML Editor `lib` folder and restart the application.

There are many navigation shortcuts which can be used for navigation in the SVG Viewer like:

- The arrow keys or **Shift Left Click** move the image.
- **Ctrl Right Click (Command Right Click on OS X)** rotates the image.
- **Ctrl I (Command I on OS X)** and **Ctrl O (Command O on OS X)** or **Ctrl Left Click (Command Left Click on OS X)** to zoom in or out.
- **Ctrl T (Command T on OS X)** to reset the transform.

The Standalone SVG Viewer

To browse and open any SVG file having the `.svg` or `.svgz` extension, use the **Tools > SVG Viewer ...** action. If the file is included in the current project, then you can open it by right-clicking on it and selecting **Open with > SVG Viewer**. The following actions are available in a contextual menu:

Zoom in

Zooms in the image by a factor of 2. The action is also available on **Mouse Wheel Up**;

Zoom out

Zooms out the image by a factor of 2. The action is also available on **Mouse Wheel Down**;

Rotate

Rotates the image 90 degrees clockwise;

Refresh

Refreshes the image, by reloading the SVG file.



Note: When you open the SVG viewer, the active SVG is displayed.

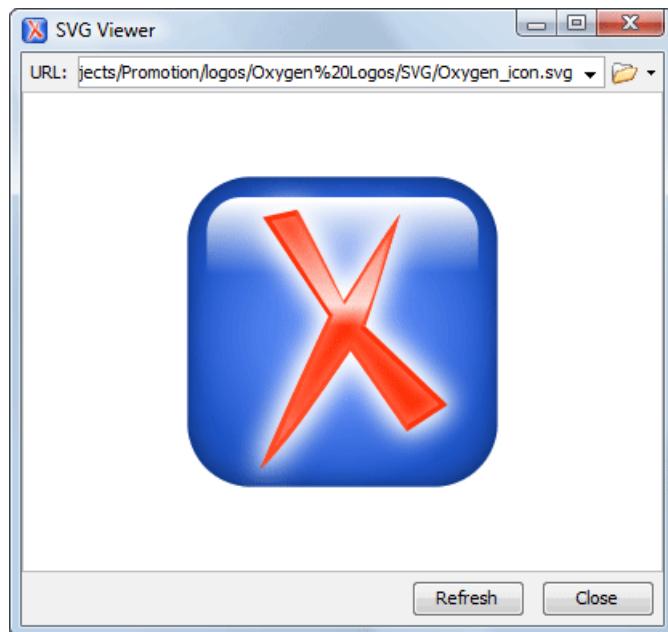


Figure 220: SVG Viewer

The Preview Result Panel

This panel can render the result of an [XSL transformation](#) that generates SVG documents.

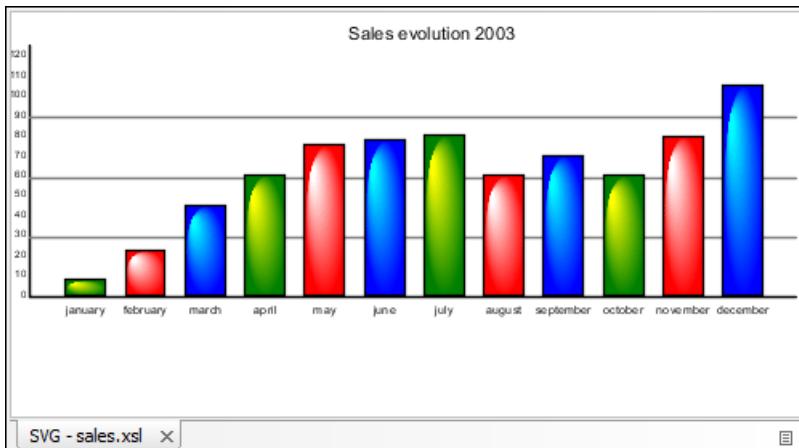


Figure 221: Integrated SVG Viewer

The basic use-case of Oxygen XML Editor consists in the development of the XSL stylesheets capable of producing rich SVG graphics. For example, you have an XML document describing the evolution of a parameter over time and you create a graphic from it. You can start with a static SVG, written directly in Oxygen XML Editor or exported from a graphics tool like the Adobe suite. Extract then the parts that are dependent of the data from the XML document and create the XSL templates. Select the option **Show as SVG** in *the dialog for configuring the XSLT transformation scenario*. When you run the transformation, the SVG result is displayed in the SVG result panel.

Editing XHTML Documents

XHTML documents with embedded CSS, JS, PHP, and JSP scripts are rendered with dedicated coloring schemes. To customize them, [open the Preferences dialog box](#) and go to **Editor > Colors**.

Spell Checking

The **Spelling** dialog allows you to check the spelling of the edited document. To open this dialog, click the **Check Spelling** toolbar button.

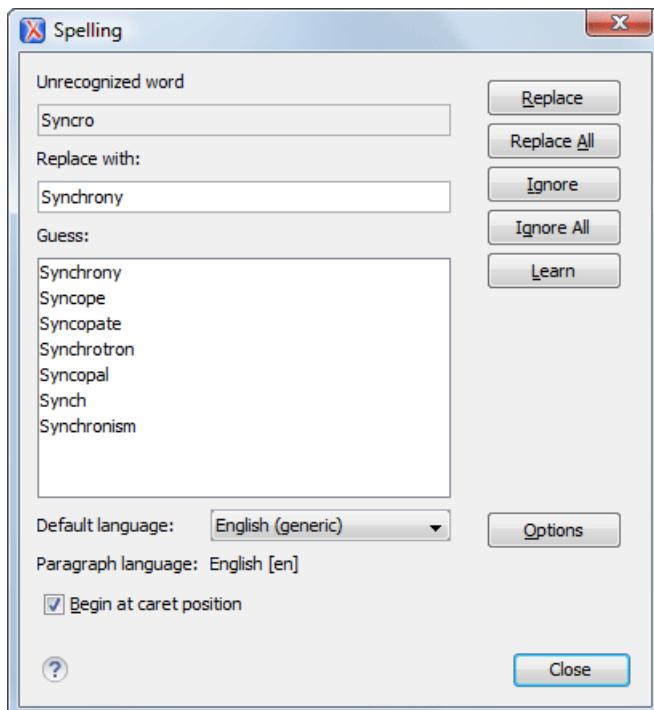


Figure 222: The Check Spelling Dialog

The dialog contains the following fields:

- **Unrecognized word** - Contains the word that cannot be found in the selected dictionary. The word is also highlighted in the XML document.
- **Replace with** - The character string which is suggested to replace the unrecognized word.
- **Guess** - Displays a list of words suggested to replace the unknown word. Double click a word to automatically insert it in the document and resume the spell checking process.
- **Default language** - Allows you to select the default dictionary used by the spelling engine.
- **Paragraph language** - In an XML document you can mix content written in different languages. To tell the spell checker engine what language was used to write a specific section, you need to set the language code in the `lang` or `xml:lang` attribute to that section. Oxygen XML Editor automatically detects such sections and instructs the spell checker engine to apply the appropriate dictionary.
- **Replace** - Replaces the currently highlighted word in the XML document, with the selected word in the **Replace with** field.
- **Replace All** - Replaces all occurrences of the currently highlighted word in the XML document, with the selected word in the **Replace with** field.
- **Ignore** - Ignores the first occurrence of the unrecognized word and allows you to continue checking the document. Oxygen XML Editor skips the content of the XML elements *marked as ignorable*.
- **Ignore All** - Ignores all instances of the unknown word in the current document.
- **Learn** - Includes the unrecognized word in the list of valid words.
- **Options** - Sets the configuration options of the spell checker.
- **Begin at caret position** - Instructs the spell checker to begin checking the document starting from the current cursor position.
- **Close** - Closes the dialog.

Spell Checking Dictionaries

There are two spell checking engines available in Oxygen XML Editor: **Hunspell** checker (default setting) and **Java** checker. You can set the spell check engine in the **Spell checking engine** preferences page. The dictionaries used by

the two engines differ in format, so you need to follow specific procedures in order to add another dictionary to your installation of Oxygen XML Editor.

Dictionaries for the Hunspell Checker

The Hunspell spell checker is open source and has LGPL license. The format of the Hunspell spell dictionary is supported by Mozilla, OpenOffice and the Chrome browser. Oxygen XML Editor comes with the following built-in dictionaries for the Hunspell checker:

- English (US)
- English (UK)
- French
- German
- Spanish.

Each language-country variant combination has its specific dictionary. If you cannot find a Hunspell dictionary that is already built for your language, you can build the dictionary you need. To build a dictionary from this list follow [these instructions](#).

Add Dictionaries and Term Lists for the Hunspell Checker

To add new spelling dictionaries to Oxygen XML Editor, or to replace an existing one, follow these steps:

1. [Download the files](#) you need for your language dictionary.
2. The downloaded .oxt file is a zip archive. If you are creating a new dictionary, copy the .aff and .dic files from this archive in the spell subfolder of the Oxygen XML Editor preferences folder.

The Oxygen XML Editor preferences folder is [APPLICATION-DATA-FOLDER]/com.oxygenxml, where [APPLICATION-DATA-FOLDER] is:

- C:\Users\[LOGIN-USER-NAME]\AppData\Roaming on Windows Vista , Windows 7, and Windows 8
- [USER-HOME-FOLDER]/Library/Preferences on OS X
- [USER-HOME-FOLDER] on Linux

3. If you are updating an existing dictionary, copy the .aff and .dic files into the folder [OXYGEN_DIR]/dicts/spell.
4. Restart the application after copying the dictionary files.



Note: You can setup Oxygen XML Editor to use dictionaries and term lists from a custom location configured in [the Dictionaries preferences page](#).

Dictionaries for the Java Checker

A Java spell checker dictionary has the form of a .dar file located in the directory [OXYGEN_DIR]/dicts. Oxygen XML Editor comes with the following built-in dictionaries for the Java checker:

- English (US)
- English (UK)
- English (Canada)
- French (France)
- French (Belgium)
- French (Canada)
- French (Switzerland)
- German (old orthography)
- German (new orthography)
- Spanish

A pre-built dictionary can be added by copying the corresponding .dar file to the folder [OXYGEN_DIR]/dicts and restarting Oxygen XML Editor. There is one dictionary for each language-country variant combination.

Learned Words

Spell checker engines rely on dictionary to decide that a word is correctly spelled. To tell the spell checker engine that an unknown word is actually correctly spelled, you need to add that word to its dictionary. There are two ways to do this:

- Press the **Learn** button from the **Spelling** dialog box.
- Invoke the contextual menu on an unknown word, then press **Learn word**.

Learned words are stored into a persistent dictionary file. Its name is composed of the currently checked language code and the `.tdi` extension, for example `en_US.tdi`. It is located in the:

- `[HOME_DIR]/Application Data/com.oxygenxml/spell` folder on Windows XP.
- `[HOME_DIR]/AppData/Roaming/com.oxygenxml/spell` folder on Windows Vista.
- `[HOME_DIR]/Library/Preferences/com.oxygenxml/spell` folder on Mac OS X.
- `[user-home-folder]/com.oxygenxml/spell` folder on Linux.



Note: To change this folder go to the [Editor > Spell Check > Dictionaries preferences page](#).



Note: To delete items from the list of learned words, press **Delete learned words** in the [Editor > Spell Check > Dictionaries preferences page](#).

Ignored Words

The content of some XML elements like `programlisting`, `codeblock` or `screen` should always be skipped by the spell checking process. The skipping can be done manually word by word by the user using the **Ignore** button of [the Spelling dialog](#) or, more conveniently, automatically by maintaining a set of known element names that should never be checked. You maintain this set of element names [in the user preferences](#) as a list of XPath expressions that match the elements.

Only a small subset of XPath expressions is supported, that is only the `'/'` and `'//'` separators and the `'*'` wildcard. Two examples of supported expressions are `/a/*/b` and `//c/d/*`.

Automatic Spell Check

To allow Oxygen XML Editor to automatically check the spelling as you write, you need to enable the **Automatic spell check** option from the [Spell Check](#) preferences page. Unknown words are highlighted and feature a contextual menu which offers the following actions:

Delete Repeated Word

Allows you to delete repeated words.

Learn Word

Allows you to add the current unknown word to the persistent dictionary.

Spell check options

Opens the [Spell Check preferences page](#).

Also, a list of words suggested by the spell checking engine as possible replacements of the unknown word is offered in the contextual menu.

Spell Checking in Multiple Files

The **Check Spelling in Files** action allows you to check the spelling on multiple local or remote documents. This action is available in:

- The **Edit** menu.
- The contextual menu of the **Project** view.
- The contextual menu of the **DITA Maps Manager** view.

The spelling corrections are displayed in [the Results view](#), that allows you to group the reported errors as a tree with two levels.

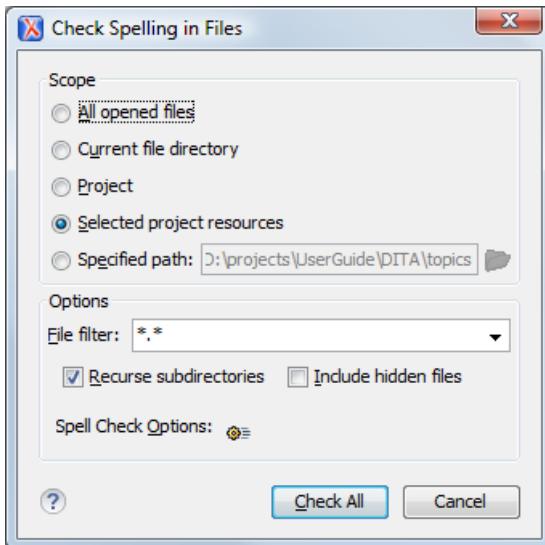


Figure 223: Check Spelling in Files Dialog

The following scopes are available:

- **All opened files** - The spell check is performed in all opened files.
- **Directory of the current file** - All the files in the folder of the current edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - Checks the spelling in the files located at a path that you specify.

The **Options** section includes the following options:

- **File filter** - Allow you to filter the files from the selected scope.
- **Recurse subdirectories** - When enabled, the spell check is performed recursively for the specified scope. The one exception is that this option is ignored if the scope is set to **All opened files**.
- **Include hidden files** - When enabled, the spell check is also performed in the hidden files.
- **Spell Check Options** - The spell check processor uses the options available in the [Spell Check preferences panel](#).

When you invoke the **Check Spelling in Files** action in the **DITA Maps Manager** view, a different dialog is displayed:

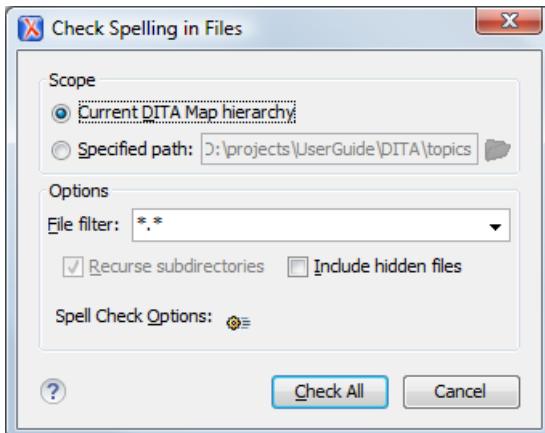


Figure 224: Check Spelling in Files Dialog in The DITA Maps Manager View

The following scopes are available:

- **Current DITA Map hierarchy** - All the files referenced in the currently selected DITA map, opened in the **DITA Maps Manager** view
- **Specified path** - checks the spelling in the files located at a path that you specify

AutoCorrect Misspelled Words

Oxygen XML Editor includes an *AutoCorrect* feature to automatically correct misspelled words, as well as to insert certain symbols or other text, as you type in **Author** mode. Oxygen XML Editor includes a default list of commonly misspelled words and symbols, but you can modify the list to suit your needs. You can also choose to have the *AutoCorrect* feature use suggestions from the main spell checker. The suggestions will only be used if the misspelled words are not found in the Replacements table.

When enabled, the *AutoCorrect* feature can be used to do the following:

- Automatically correct misspelled words while you edit in **Author** mode.
- Easily insert symbols. For example, if you want to insert a ® character, you would type (R).
- Quickly insert text fragments.

To enable and configure this feature, [open the Preferences dialog box](#) and go to **Editor > Edit Modes > Author > AutoCorrect**.

The *AutoCorrect* feature results in the following types of substitutions in regards to case-sensitivity:

- Words with all lower-case characters will be replaced with lower-case substitutions (for example, "abotu" is replaced with "about").
- Words with irregular-case characters will be replaced with lower-case substitutions ("ABotU" is replaced with "about").
- Words with all upper-case characters will be replaced with upper-case substitutions ("ABOTU" is replaced with "ABOUT").
- Words starting with an upper-case character will be replaced with substitutions having the same pattern ("Abotu" is replaced with "About").

The actual operation of replacing of a word is triggered by a space, dash, or punctuation mark (, . ; : ? !).

The *AutoCorrect* feature also uses the list of [ignored elements from the Spell Check preferences page](#). All elements (along with their descendant elements) included in this list will be ignored by the *AutoCorrect* engine.

Add Dictionaries for the AutoCorrect Feature

To add new dictionaries for the *AutoCorrect* feature, or to replace an existing one, follow these steps:

1. [Download the files](#) you need for your language dictionary.
2. If you are creating a new dictionary, copy the downloaded .dat files to the `autocorrect` subfolder of the Oxygen XML Editor preferences folder.

The Oxygen XML Editor preferences folder is `[APPLICATION-DATA-FOLDER]/com.oxygenxml`, where `[APPLICATION-DATA-FOLDER]` is:

- `C:\Users\[LOGIN-USER-NAME]\AppData\Roaming` on Windows Vista, Windows 7, and Windows 8
- `[USER-HOME-FOLDER]/Library/Preferences` on OS X
- `[USER-HOME-FOLDER]` on Linux

3. If you are updating an existing dictionary, copy the .dat file to the folder `[OXYGEN_DIR]/dicts/autocorrect`.
4. Restart the application after copying the dictionary files.



Note: You can setup Oxygen XML Editor to use dictionaries from a custom location configured in [the Dictionaries preferences page](#).

Editing Large Documents

When you open a document with a file size larger than the limit configured in [Open/Save preferences](#), Oxygen XML Editor prompts you to choose whether you want to optimize the loading of the document for large files or for huge files.

If your file has a size smaller than 300 MB, the recommended approach is [Optimize loading for large files](#). For documents that exceed 300 MB the recommended approach is [Optimize loading for huge files](#).

File sizes smaller than 300 Megabytes

For editing large documents (file size up to 300 Megabytes), a special memory optimization is implemented on loading such a file so that the total memory allocated for the application is not exceeded.

A temporary buffer file is created on disk so you have to make sure that the available free disk space is at least double the size of the large file that you want to edit. For example, Oxygen XML Editor can load a 200-Megabytes file using a minimum memory setting of 512 Megabytes and at least 400-Megabytes free disk space.

The increase of the maximum size of editable files comes with the following restrictions:

- A file larger than the value of the above option is edited only in Text mode.
- The [automatic validation](#) is not available when editing a very large file.
- The XPath filter is disabled in [the Find/Replace dialog box](#).
- The bidirectional Unicode support (right-to-left writing) is disabled.
- [The option Format and indent the document on open](#) is disabled for non-XML documents. For XML documents, it is done optimizing the memory usage but without respecting the options set in [the Format preferences page](#).
- Less precise localizations for the results of an [XPath expression](#).

File sizes greater than 300 MB

Files tend to become larger and larger mostly because they are frequently used as a format for database export or for porting between different database formats. Traditional text editors simply cannot handle opening these huge export files, some having sizes exceeding one gigabyte, because all the file content must be loaded in memory before the user can actually view it.

The file is split in multiple pages (each having about 1MB in size). Each page is individually loaded (and edited) in the **Text** mode by using the special horizontal slider located at the top of the editing area. The loading operation is very fast and has no upper limit for the size of the loaded file.

The increase of the maximum size of editable files comes with the following restrictions:

- For XML files, only the UTF-8, UTF-16, and ASCII encodings are handled; for all non-XML files, the content is considered to be UTF-8.
- Files can be edited in Text edit mode only.
- The [automatic validation](#) is disabled.
- The XPath filter is disabled in [the Find/Replace dialog box](#).
- The bidirectional Unicode support (right-to-left writing) is disabled.
- [The Format and indent the document on open option](#) is disabled for non-XML documents. For XML documents, the format and indent operation it is done optimizing the memory usage, but it ignores the options set in [the Format preferences page](#).
- The **Outline** view is not supported.
- The file content is soft wrapped by default.
- The **Find/Replace** dialog box only supports the **Find** action.
- Saving changes is possible if **Safe save** is activated.

- The **undo** operation is not available if you go to other pages and come back to the modified page. the Undo operation loses its previous states if the back and forth between

Scratch Buffer

A handy addition to the document editing is the **Scratch Buffer** view used for storing fragments of arbitrary text during the editing process. It can be used to drop bits of paragraphs (including arbitrary XML markup fragments) while rearranging and editing the document and also to drag and drop fragments of text from the scratch buffer to the editor panel. The **Scratch Buffer** is basically a text area offering XML syntax highlight. The view contextual menu contains basic edit actions like **Cut**, **Copy**, and **Paste**.

Handling Read-Only Files

If a file marked as read-only is opened in Oxygen XML Editor you can by default perform modifications to it. This behavior is controlled by the [Can edit read only files](#) option. When attempting to save such files you will be prompted to save them to another location.

You can check out the read-only state of the file by looking in the [Properties view](#). If you modify the file properties from the operating system and the file becomes writable, you are able to modify it on the spot without having to reopen it.

The read-only state is marked with a lock decoration which appears in the editor tab and specified in the tooltip for a certain tab.

Editing Documents with Long Lines

The documents containing long lines can affect performance when opened in the **Text** mode. If you choose to present the document with line wrap, some features are affected:

- The editor uses the Monospaced font.
- You cannot set font styles.
- Automatic validation is disabled.
- Automatic spell checking is disabled.
- **XPath** field is disabled in the **Find/Replace** dialog box.
- Less precise localization for executed XPaths. The XPath executions use SAX sources for smaller memory footprint. We recommend using XPath 2.0 instead of XPath 1.0 because it features an increased execution speed and uses a smaller memory footprint. Running an XPath expression requires additional memory about 2 or 3 times the size of the document on disk.

The last two restrictions are valid only for XML documents.

Associating a File Extension with Oxygen XML Editor

To associate a file extension with Oxygen XML Editor on Windows:

- Go to the **Start** menu and click **Control Panel**.
- Go to **Default Programs**.
- Click **Associate a file type or protocol with a program**.
- Click the file extension you want to associate with Oxygen XML Editor, then click **Change program**.
- In the **Open With** dialog box, click **Browse** and navigate to Oxygen XML Editor.

To associate a file extension with Oxygen XML Editor on Mac:

- In **Finder**, right click a file and from the contextual menu select **Get Info**.

- In the **Open With** subsection, select **Other** from the application combo and browse to Oxygen XML Editor.
- With Oxygen XML Editor selected, click **Change All**.

Chapter

8

Author for DITA

Topics:

- [*Creating DITA Maps and Topics*](#)
- [*DITA Maps Manager*](#)
- [*Transforming DITA Maps and Topics*](#)
- [*DITA-OT Customization*](#)
- [*DITA Specialization Support*](#)
- [*Use an External DITA Open Toolkit in Oxygen XML Editor*](#)
- [*Reusing Content*](#)
- [*Moving and Renaming Resources*](#)
- [*DITA Profiling / Conditional Text*](#)
- [*Working with MathML*](#)

This chapter presents the Author features that are specific for editing DITA XML documents.

Creating DITA Maps and Topics

The basic building block for DITA information is the DITA topic. DITA provides a number of different topic types, the most common of which are:

- *Concept* - For general, conceptual information such as a description of a product or feature.
- *Task* - For procedural information such as how to use a dialog box.
- *Reference* - For reference information.

You can organize topics into a *DITA map* or *bookmap*.

DITA Maps Manager

Oxygen XML Editor provides a view for managing and editing *DITA Maps*. The **DITA Maps Manager** view presents a DITA map as a table-of-contents. It allows you to navigate to the topics and maps, make changes, and apply transformation scenarios to obtain various output formats.

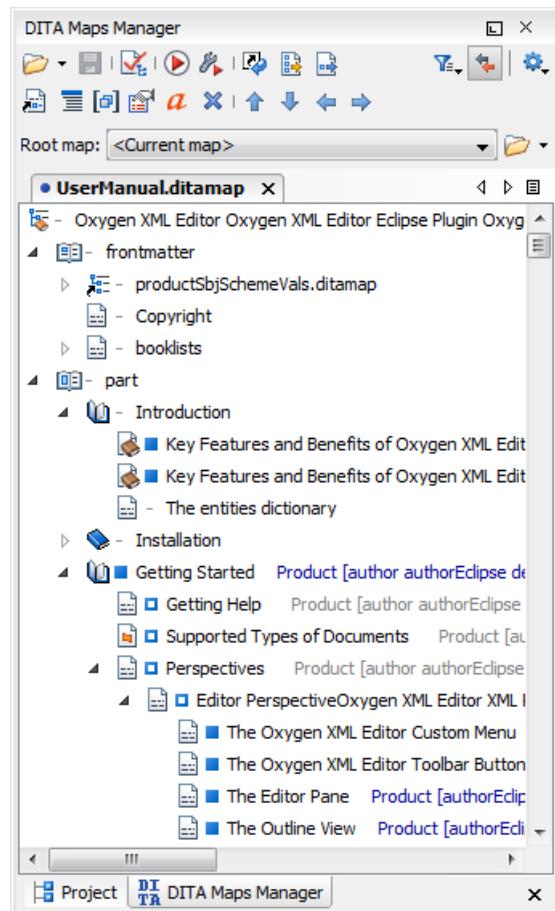


Figure 225: The DITA Maps Manager View

The **DITA Maps Manager** view supports multiple open maps at the same time.

When you open a file with the extension `.ditamap` or `.bookmap`, Oxygen XML Editor offers you the choice of opening it in the XML editor or in the **DITA Maps Manager**. In addition, you can right click a map file in the **Project** view and select **Open with**. If you have a ditamap file open in the XML editor, you can open it in the **DITA Maps Manager** by right-clicking on the title tab and selecting the **Open in DITA Maps Manager**.

If your map references other DITA Maps, they will be shown, expanded, in the DITA Maps tree and you will be able to navigate their content. To edit them you need to open each referenced map in a separate editor. You can choose not to expand referenced maps in the **DITA Maps Manager** view, or referenced content in the opened editors, by unchecking the **Display referenced content** checkbox available in the [Author preferences page](#).

Drag and Drop in the DITA Maps Manager

You can move topics in the same map, or between different maps, by dragging and dropping them into the desired position. Also, you can move multiple topics by dragging them while pressing the **Ctrl (Command on OS X)** key.

You can also arrange the nodes by dragging and dropping one or more nodes at a time. Drop operations can be performed before, after, or as child of the targeted node. The relative location of the drop is indicated while hovering the mouse over a node before releasing the mouse button for the drop.

Drag and drop operations include:

Copy

Select the nodes you want to copy and start dragging them. Before dropping them in the appropriate place, press and hold the **Ctrl** key (**Meta** key on Mac). The mouse pointer changes to indicate that a copy operation is performed.

Move

Select the nodes you want to move and drag and drop them in the appropriate place.

Promote (Alt Left Arrow)/Demote (Alt Right Arrow)

You can move nodes between child and parent nodes by using the **Promote (Alt Left Arrow)** and **Demote (Alt Right Arrow)** operations.

DITA Maps Manager Toolbar

The toolbar includes the following actions (also available in the **DITA Maps** menu):

Open

You can use this drop-down menu to reopen recently viewed DITA maps. The drop-down menu also contains the following actions:

- **Clear history** - Clears the history list of the recently viewed DITA maps.
-  **Open...** - Allows you to open the map in the **DITA Maps Manager** view. You can also open a map by dragging it from the file system explorer and dropping it into the **DITA Maps Manager** view.
-  **Open URL...** - Displays the **Open URL** dialog box that allows you to access any resource identified through a URL (defined by a protocol, host, resource path, and an optional port). The following actions are available in this drop-down action list:
 -  **Browse for local file** - Opens a local file browser dialog box, allowing you to select a local DITA map.
 -  **Browse for remote file** - Displays [the Open URL dialog box](#) that allows you to open a remotely stored DITA map.
 -  **Browse for archived file** - Displays the **Archive Browser** dialog box that allows you to browse the content of an archive and choose a DITA map.
 -  **Browse Data Source Explorer** - Opens the **Data Source Explorer** that allows you to browse the data sources defined in the [Data Sources preferences page](#).
-  **Tip:** You can open the **Data Sources** preferences page by using the **Configure Database Sources** shortcut from the **Open URL** dialog box.
-  **Search for file** - Displays the [Open/Find Resource dialog box](#).

Save (**Ctrl (Meta on Mac OS)+S**)

Saves the current DITA map.

 **Validate and Check for Completeness**

Checks the validity and integrity of the map.

 **Apply Transformation Scenario(s)**

Applies the DITA Map transformation scenario that is associated with the current map.

 **Configure Transformation Scenario(s)**

Allows you to *associate a DITA Map transformation scenario* with the current map.

 **Refresh References**

You can use this action to manually trigger a refresh and update of all referenced documents. This action is useful when the referenced documents are modified externally. When they are modified and saved from the Oxygen XML Editor Author, the DITA map is updated automatically.

 **Open Map in Editor with Resolved Topics**

Opens the DITA map in the main editor area with content from all topic references, expanded in-place. Content from the referenced topics is presented as read-only and you have to use the contextual menu action **Edit Reference** to open the topic for editing.

 **Tip:** If you want to print the expanded content, you should consider changing the **Styles** drop-down to + **Print ready**.

 **Open Map in Editor**

For complex operations that cannot be performed in the simplified **DITA Maps Manager** view (for instance, editing a relationship table) you can open the map in the main editing area.

 **Note:** You can also use this action to open referenced DITA maps in the **Editor**.

 **Profiling/Conditional Text**

This drop-down list contains the following actions:

- **Show Profiling Colors and Styles** - Enable this option to turn on conditional styling. To configure the colors and styles *open the Preferences dialog box* and go to **Editor > Edit modes > Author > Profiling/Conditional Text > Colors and Styles**.
- **Show Profiling Attributes** - Enable this options to display the values of the profiling attributes at the end of the titles of topic references. When enabled, the values of the profiling attributes are displayed in both the **DITA Maps Manager** view and in the **Author** view.
- **Show Excluded Content** - Controls if the content filtered out by a particular condition set is hidden or greyed-out in the editor area and in the **Outline** and **DITA Maps Manager** views. When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.
-  **Profiling Settings** - Opens the preferences page for adding and editing the profiling conditions that you can apply in the **DITA Maps Manager** view and the **Author** view. When a profiling condition set is applied, the keys that are defined in the DITA map are gathered by filtering out the excluded content.

 **Link with Editor**

Disables/Enables the synchronization between the file path of the current editor and the selected topic reference in the **DITA Maps Manager** view.

 **Note:** This button is disabled automatically when you move to the **Debugger** perspective.

 **Settings**

Allows you to choose whether or not to **Show extended toolbar** and **Show root map toolbar**.

Root map

Specifies a master DITA map that Oxygen XML Editor uses to establish a *key space* that you can use with any other DITA map that is contained by the master map.

Contextual Menu of the DITA Maps Manager

The following actions can be invoked from the contextual menu on the *root map* of an opened DITA Map:



Open Map in Editor

For complex operations that cannot be performed in the simplified DITA Maps view (for instance, editing a relationship table) you can open the map in the main editing area.



Open Map in Editor with Resolved Topics

Opens the DITA map in the main editor area with content from all topic references, expanded in-place. Content from the referenced topics is presented as read-only and you have to use the contextual menu action **Edit Reference** to open the topic for editing.

Export DITA Map...

Allows you to choose a destination for exporting the DITA map.

Find Unreferenced Resources...

Allows you to search for orphaned resources that are not referenced in the DITA maps.



Edit Attributes...

Allows you to edit all the attributes of a selected node. You can find more details about this action in the [Attributes View](#) on page 121 topic.

Edit Profiling Attributes...

Allows you to change the *profiling attributes* defined on all selected elements.



Edit Properties...

Edit the properties of a selected node. You can find more details about this action in the [Edit Properties in DITA Maps](#) on page 444 topic.

Append Child

Container sub-menus for a number of actions that create a map node as a child of the currently selected node, or as a sibling of the currently selected node:

- **New topic...** - Inserts a new topic.
- **Reference...** - Inserts a reference to a topic file. You can find more details about this action in the [Inserting References](#) topic.
- **Reference to the currently edited file...** - Inserts a reference to the currently edited file.
- A set of actions that allow you to insert various *reference specializations* (such as **Anchor Reference**, **Key Definition**, **Map Reference**, **Topic Group**, **Topic Head**, **Topic Reference**, **Topic Set**, **Topic Set Reference**).
- **Topic Heading...** - Inserts a topic heading. You can find more details about this action in the [Inserting Topic Headings](#) topic.
- **Topic Group...** - Inserts a topic group. You can find more details about this action in the [Inserting Topic Groups](#) on page 444 topic.

Search References

Searches all references to the current topic in the entire ditamap.

Refactoring > Rename resource...

Allows you to *change the name of a resource linked in the edited DITA map*.

Refactoring > Move resource...

Allows you to *change the location on disk of a resource linked in the edited DITA map*.

Refactoring >  XML Refactoring...

Opens [the XML Refactoring tool wizard](#) that presents refactoring operations to assist you with managing the structure of your XML documents.

Find/Replace in Files...

Allows you to find and replace content across multiple files.

Check Spelling in Files...

Allows you to [spell check multiple files](#).

Paste

Allows you to paste content from the clipboard into the DITA map.

Paste Before

Pastes the content of the clipboard (only if it is a part of the DITA map) before the currently selected DITA map node.

Paste After

Pastes the content of the clipboard (only if it is a part of the DITA map) after the currently selected DITA map node.

Expand All

Allows you to expand the entire DITA map structure.

Collapse All

Allows you to collapse the entire DITA map structure.

In addition to those described above, the following actions are available when the contextual menu is invoked from child nodes of the *root map*:

Open

Opens in the editor the resources referenced by the nodes that you select.

Cut, Copy, Paste, Delete

Common edit actions that allow you to cut, copy, paste, and delete parts of the DITA map.

Organize

Allows you to organize the DITA map with the several submenu actions:

-  **Move Up** - moves the selected node up within the DITA map tree.
-  **Move Down** - moves the selected node down within the DITA map tree.
-  **Promote (Alt Left Arrow)** - moves the selected node up one level to the level of its parent node.
-  **Demote (Alt Right Arrow)** - moves the selected node down one level to the level of its child nodes.

To watch our video demonstrations about DITA editing and the **DITA Maps Manager** view, go to

http://oxygenxml.com/demo/DITA_Editing.html and http://oxygenxml.com/demo/DITA_Maps_Manager.html, respectively.

Creating a Map

To create a DITA map, Subject scheme, bookmap, or other types of DITA maps, follow these steps:

1. Go to **File > New**.
A **New** document dialog box is opened that allows you to select a document type from various folders.
2. Select one of the **DITA Map** templates from the **Framework templates** folder.
3. Click the **Create** button.
4. Select whether you want to open the map in the **DITA Maps Manager** or the **Editor**.
5. Save the map using the  **Save** button on the toolbar of the **DITA Maps Manager** view.

Selecting a Root Map

Oxygen XML Editor allows you to select a DITA Map as a *key space*, or *root map*, for all the other DITA Maps and topics in the project. Specifying the correct *root map* helps to prevent validation problems when you work with `keyrefs` and also acts as the foundation for content completion. All the *keys* that are defined in a *root map* are available in the maps that the *root map* contains.

There are several ways to select or change the *root map*:

- Use the **Root map** drop-down lists in the **DITA Maps Manager** toolbar to select the appropriate *root map*.
- From the **DITA** toolbar or contextual menu select **Link > Key Reference...** to open the **Insert Key Reference** dialog and click on the **Change Root Map** link at the top of the dialog.
- From the **DITA** toolbar click the **Insert Content Key Reference** button to open the **Insert Content Key Reference** dialog and click on the **Change Root Map** link at the top of the dialog.

Note: You can also click a key reference error to select the root map.

To watch our video demonstration about the DITA Root Map support, go to
http://oxygenxml.com/demo/DITA_Root_Map.html.

Create a Topic in a Map

To add a topic to a DITA map:

1. Select a node of a map open in the **DITA Maps Manager View**.
2. To insert the topic as a child of the selected node, right click that node and choose **Insert > Append Child**. To insert the topic as a sibling to the current node, choose **Insert > Insert After**. Then select the type of reference you want to create.
 The **Insert Reference** dialog box is displayed.
3. Select the topic to insert and press the **Insert** button or the **Insert and close** button.
 A reference to the selected topic is added to the current map in the view.
4. If you clicked the **Insert** button you can continue inserting new topic references using the **Insert** button repeatedly.
5. Close the dialog box by using the **Close** button.

Organize Topics in a Map

To understand how to organize topics in a *DITA map* using the **DITA Maps Manager**, you can examine the sample map called `flowers.ditamap`, located in the `[OXYGEN_DIR]/samples/dita` folder.

1. Open the file `flowers.ditamap`.
2. Select the gear icon in the top right corner of the **DITA Maps Manager** and select **Show extended toolbar**.
3. Select the topic reference *Summer Flowers* and press the **Move Down** button to change the order of the topic references *Summer Flowers* and *Autumn Flowers*.
4. Make sure that *Summer Flowers* is selected and press the **Demote** button. This topic reference and all the nested ones are moved as a unit inside the *Autumn Flowers* topic reference.
5. Close the map without saving.

Creating Relationship Tables

You can define relationships between topics in a relationship table. A relationship table is created inside a *DITA map*.

1. If the map is currently open in the **DITA Maps Manager**, double-click the map icon to open the map in **Author** mode. If it opens in **Text** mode, click **Author** at the bottom left to switch to **Author** mode.
2. Go to **DITA > Relationship Table > Insert Relationship Table**.
 The **Insert Relationship Table** dialog box is displayed.

3. Set the number of rows, the number of columns, a table title (optional), and select whether you want a table header. Click **Insert**.
4. Enter the type of the topics in the header of each column.

The header of the table (the `relheader` element) already contains a `relcolspec` element for each table column. You should set the value of the attribute `type` of each `relcolspec` element to a value like `concept`, `task`, `reference`. When you click in the header cell of a column (that is a `relcolspec` element), you can see all the attributes of that `relcolspec` element, including the `type` attribute in the **Attributes** view. You can edit the attribute type in this view.

5. To insert a topic reference in a cell, place the cursor in a table cell and click  **Insert Reference** from the contextual menu or the **DITA Map** toolbar.
6. To add a new row to the table or remove an existing row use  **Insert Relationship Row**/ **Delete Relationship Row** from the contextual menu or the **DITA Map** toolbar.
7. To add a new column to the table or remove an existing column, use  **Insert Relationship Column**/ **Delete Relationship Column** contextual menu or the **DITA Map** toolbar. If you double-click the relationship table (or select it and press **Enter**, or choose **Open** from the contextual menu) the DITA map is opened in the editor with the caret positioned inside the corresponding relationship table.

 **Note:** When the map is open in the **DITA Maps Manager**, the newly created relationship table is also displayed there. If you double-click on the relationship table (or select it and press **Enter**, or choose **Open** from the contextual menu) the DITA map will be opened in the editor with the caret positioned inside the corresponding relationship table.

Validating DITA Maps

To validate a DITA map, go to the *the DITA Maps Manager view* and click  **Validate and Check for Completeness**. You can also find the  **Validate and Check for Completeness** action in the **DITA Maps** menu. Invoking this action opens the **DITA Map completeness Check** dialog box, which allows you to configure the DITA Map validation.

The validation process of a DITA MAP covers the following steps:

- verifies whether the file paths of the topic references are valid. In case an `href` attribute points to an invalid file path it is reported as a separate error in the **Errors** view.
- validates each referenced topic and map. Each topic file is opened and validated against the appropriate DITA DTD. In case another DITA map is referenced in the main one, the referenced DITA Map is verified recursively, applying the same algorithm as for the main map.

The following options are available in the **DITA Map Completeness Check** dialog box:

- **Batch validate referenced DITA resources** - this option decides the level of validation that applies to referenced DITA files:
 - if the check box is left unchecked (which is the default setting), the DITA files will be validated using the rules defined in the DTD or XML Schema declared in the document.
 - if the check box is checked, the DITA files will be validated using rules defined in their associated *validation scenario*.
- **Check the existence of non-DITA references resources** - extends the validation of referenced resources to non-DITA files. Enable the **Include remote resources** options if you want to check that remote referenced binary resources (like images, movie clips, ZIP archives) exist at the specified location.
- **Use DITAVAL filters** - the content of the map is filtered by applying a *profiling condition set* before validation:
 - **From the current condition set** - the map is filtered using the condition set applied currently in the DITA Maps Manager view.
 - **From all available condition sets** - for each available condition set, the map content is filtered using the condition set before validation.

- **From the associated transformation scenario** - the filtering condition set is specified explicitly as a DITAVAL file in the current transformation scenario associated with the DITA map.
- **Other DITAVAL files** - for each DITAVAL file from this list, the map content is filtered using the DITAVAL file before validation.



Note: A link invalid in the content that resulted from the filtering process is reported as an error.

- **Check for duplicate topic IDs within the DITA map context** - checks for multiple topics with the same ID in the context of the entire map.
- **Report links to topics not referenced in DITA maps** - checks that all referenced topics are linked in the DITA map.
- **Identify possible conflicts in profile attribute values** - when a topic's profiling attributes contain values that are not found in parent topics profiling attributes, the content of the topic is overshadowed when generating profiled output. This option reports such possible conflicts.
- **Report attributes and values that conflict with profiling preferences** - looks for profiling attributes and values not defined in the *Profiling / Conditional Text* preferences page. It also checks if profiling attributes defined as *single-value* have multiple values set in the searched topics.
- **Additional schematron checks** - allows you to select a Schematron schema that Oxygen XML Editor uses for the validation of DITA resources.

Finding Resources Not Referenced in DITA Maps

Over the course of time large projects can accumulate a vast amount of resources from a variety of sources. Especially in organizations with a large number of content writers or complex project structures, organizing the project resources can become a challenge. Over time a variety of actions can cause resources to become orphaned from DITA maps. To assist you with organizing project resources, Oxygen XML Editor includes an action, **Find Unreferenced Resources**, that searches for orphaned resources that are not referenced in DITA maps.

To perform this search, open the DITA map in the **DITA Maps Manager**, invoke the contextual menu on the DITA map, and select **Find Unreferenced Resources**. This action can also be selected from the **DITA Maps** menu. This action opens the **Find Unreferenced Resources** dialog box, which allows you to specify some search parameters:

- **DITA Maps** - Provides a list of DITA maps to be included in the search and allows you to **Add** maps to the list or **Remove** them.
- **Folders** - Provides a list of folders to be included in the search and allows you to **Add** or **Remove** specific folders.
- **Filters** - Provides three combo boxes that allow you to filter the search to include or exclude certain files or folders:
 - **Include files** - Allows you to filter specific files to include in the search.
 - **Exclude files** - Allows you to filter specific files to exclude from the search.
 - **Exclude folders** - Allows you filter specific folders to exclude from the search.



Note: In any of the filter combo boxes you can enter multiple filters by separating them with a comma and you can use the ? and * wildcards. Use the drop-down arrow to select a previously used filter pattern.

Insert and Edit References

This section explains how to insert and edit references (such as topic references, topic groups, topic headings, and key definitions) in a DITA map.

Inserting References

A DITA map may contain various types of references. The targets of the references can be a variety of different references, such as anchors, chapters, maps, topics, or topic sets.

You can insert references to targets such as anchors, topics, maps, topic sets, or key definitions with the **Insert Reference** dialog box. This dialog box can be opened from the **DITA Maps Manager** extended toolbar or with *actions from the contextual menu in the DITA Maps Manager view* (using the **Append child** and **Insert after** submenus).

The content of these submenus depends on the node that is selected in the DITA map tree when the contextual menu is invoked. For example, if the selected node is a topic reference (`topicref`), its possible child nodes include the following elements: `anchorref`, `chapter`, `keydef`, `mapref`, `topicgroup`, `topichead`, `topicref`, `topicset`, and `topicsetref`.

Open the **Insert Reference** dialog box by using the  **Insert Reference** button on the toolbar or from the contextual menu (**Append child** >  **Reference...** or **Insert after** >  **Reference...**).

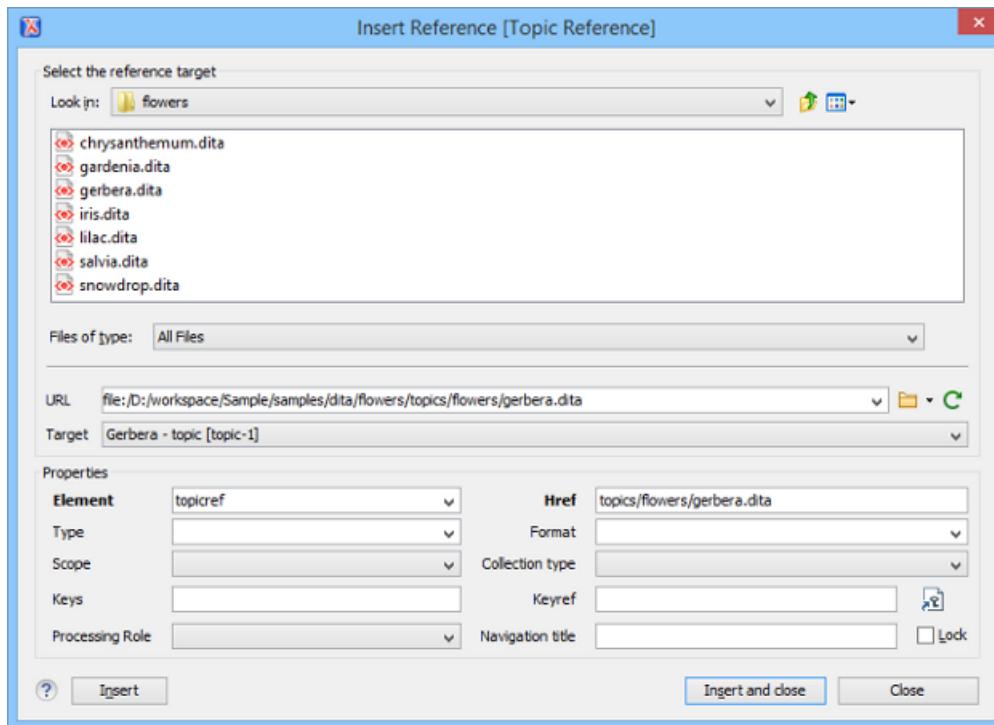


Figure 226: Insert Reference Dialog Box

The **Insert Reference** dialog box offers the following sections and actions:

Select the reference target

Using the browse tools, file window, and filter tool in this section, you can easily browse for and select the source target file.

Target

The **URL** combo box specifies the path to the target that holds the content you want to reference and the **Target** drop-down list shows all available target.

Element

You can use this combo box to specify the reference element.

Href

The selected target automatically modifies this value to point to the corresponding `href` attribute of the inserted `topicref` element.

Type

Allows you to select a `type` attribute (such as `topic`, `task`, `concept`, etc.) of the inserted element.

Format

This property is filled automatically, based on the selected file, and corresponds to the `format` attribute of the inserted element.

Scope

This property is filled automatically, based on the selected file, and corresponds to the `scope` attributes of the inserted element.

Collection type

Drop-down list that allows you to select the `collection-type` attribute to create hierarchical linking between topics in a DITA map (for example `unordered`, `sequence`, `choice`, `family`, `-dita-use-conref-target`).

Keys

Use this text field to define the `keys` attribute on the inserted reference.

Keyref

Instead of using the **Href** combo box to point to a location you can reference a key definition by using this text field. Use the  **Choose key reference** button to access the list of keys defined in the currently opened DITA map.

Processing Role

This drop-down list allows you to set the `processing-role` attribute to one of the allowed values for DITA reference elements (for example `resource-only`, `normal`, `-dita-use-conref-target`).

Navigation title

This text field allows you to specify a custom navigation title for the inserted reference and to enforce it by using the **Lock** checkbox.

Once you click **Insert** or **Insert and close**, the selected target will be added as a child or sibling of the selected reference, depending on the insert action selected from the contextual menu of the **DITA Maps** view (**Append child** or **Insert after**).

-  **Note:** You can easily insert multiple topic references by keeping the dialog box opened and changing the selection in the **DITA Maps Manager** tree. You can also select multiple resources in the file explorer and then insert them all as topic references.
-  **Tip:** Another way to easily insert a reference is to drag files from the **Project** view, file system explorer, or **Data Source Explorer** view and drop them into the map tree.

Inserting Topic Headings

The `topichead` element provides a title-only entry in a navigation map, as an alternative to the fully-linked title provided by the `topicref` element.

A topic heading can be inserted both from the toolbar action and the contextual node actions.

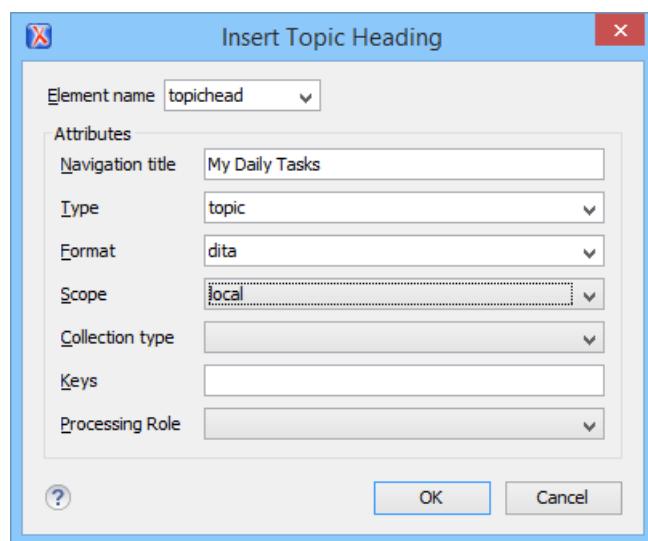


Figure 227: Insert Topic Heading Dialog Box

By using the **Insert Topic Heading** dialog box you can easily insert a `topichead` element. The **Navigation title** is required but other attributes can also be specified from this dialog box.

Inserting Topic Groups

The `topicgroup` element identifies a group of topics (such as a concepts, tasks, or references) or other resources. A `topicgroup` can contain other `topicgroup` elements, allowing you to express navigation or table-of-contents hierarchies, as well as implying relationships between the containing `topicgroup` and its children. You can set the collection-type of a container `topicgroup` to determine how its children are related to each other. Relationships end up expressed as links in the output (with each participant in a relationship having links to the other participants by default).

A topic group may be inserted both from the toolbar action and the contextual node actions.

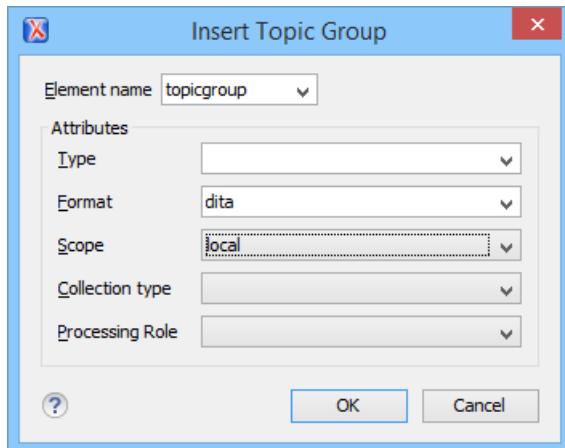


Figure 228: Insert Topic Group Dialog Box

By using the **Insert Topic Group** dialog box, you can easily insert a `topicgroup` element. The **Type**, **Format**, **Scope**, and **Collection type** attributes can be specified from this dialog box.

Edit Properties in DITA Maps

The **Edit properties** action, available both on the toolbar and on the contextual menu, is used to edit the properties of the selected node. Depending on the selected node, the action will perform the following tasks:

- If a `topicref` or `chapter` element is selected, the action opens a dialog box that is similar to the [Insert Topic Reference dialog box](#), allowing you to edit some of the important attributes.
- If a `topichead` element is selected, the action opens a dialog box that is similar to the [Insert Topic Heading dialog box](#), allowing you to edit some of the important attributes.
- If a `topicgroup` element is selected, the action opens a dialog box that is similar to the [Insert Topic Group dialog box](#), allowing you to edit some of the important attributes.
- If the root element of the map is selected, you can easily edit the map title by using the **Edit Map title** dialog box. Using this dialog box, you can also specify if the title is specified as the `title` attribute for the map, as a `title` element (for DITA-OT 1.1 and 1.2), or specified in both locations.

Transforming DITA Maps and Topics

Oxygen XML Editor uses the DITA Open Toolkit (DITA-OT) to transform DITA maps and topics into an output format. For this purpose both the DITA Open Toolkit and ANT come bundled in Oxygen XML Editor.

More information about the DITA Open Toolkit are available at <http://dita-ot.sourceforge.net/>.

DITA OT Transformation

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**.
 - Use the **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **DITA OT Transformation**.
 - Use the **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **DITA OT Transformation**.
- Note:** If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **DITA Transformation Type** dialog box that presents the list of possible outputs.

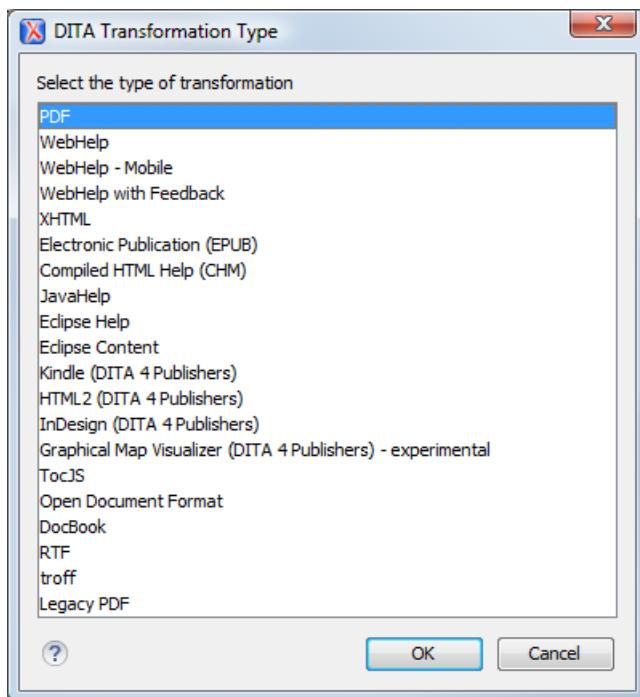


Figure 229: DITA Transformation Type Dialog Box

Select the desired type of output and click **OK**. This opens the **New Scenario** dialog box, which allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs (only those that are appropriate for the chosen output type will be displayed):

- **Skins** (Available for **WebHelp** and **WebHelp with Feedback** output types).
- **FO Processor** (Available for **PDF** output types).

- [Parameters](#)
- [Filters](#)
- [Advanced](#)
- [Output](#)

For information on creating an entirely new DITA OT transformation, see [Creating a DITA OT Customization Plugin](#) on page 456 and [Installing a Plugin in the DITA Open Toolkit](#) on page 457.

The FO Processor Tab

This tab allows you to select an FO Processor, when you choose to generate PDF output.

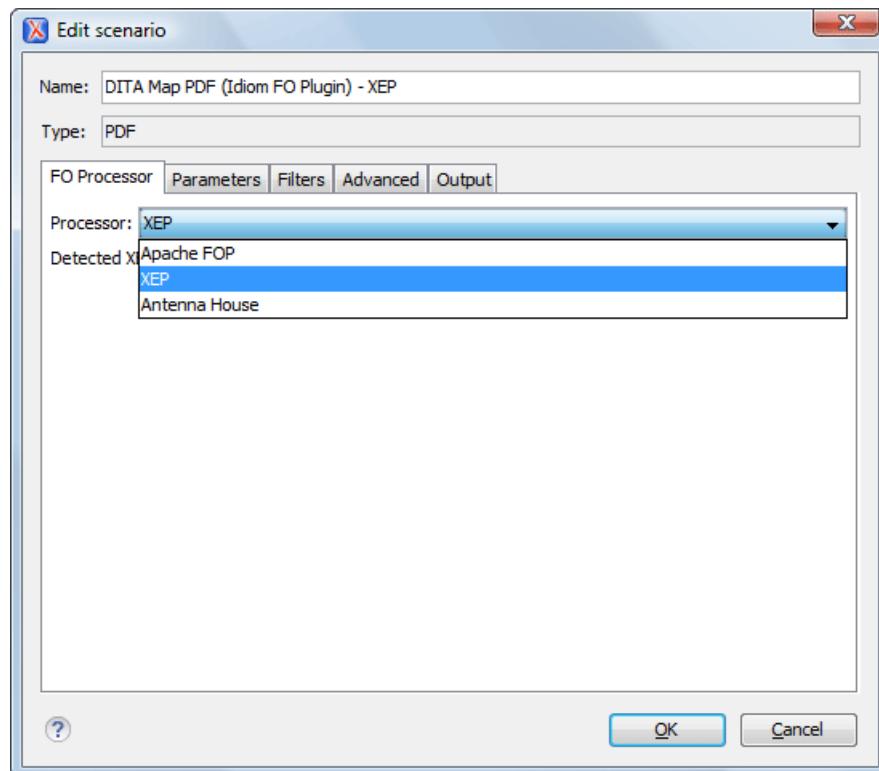


Figure 230: FO Processor Configuration Tab

You can choose the following processors:

- **Apache FOP** - The default processor that comes bundled with Oxygen XML Editor.
- **XEP** - The [RenderX](#) XEP processor.

If XEP is already installed, Oxygen XML Editordisplays the detected installation path under the drop-down list.

XEP is considered installed if it was detected in one of the following sources:

- XEP was configured as an external FO Processor in the [FO Processors option page](#).
- The system property `com.oxygenxml.xep.location` was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows).
- XEP was installed in the
[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/lib directory of the Oxygen XML Editor installation directory.
- **Antenna House** - The [Antenna House](#) AH (v5) or XSL (v4) Formatter processor.

If Antenna House is already installed, Oxygen XML Editordisplays the detected installation path under the drop-down list.

Antenna House is considered installed if it was detected in one of the following sources:

- Environment variable set by Antenna House installation (the newest installation version will be used, v5 being preferred over v4).
- Antenna House was added as an external FO Processor in the Oxygen XML Editor preferences pages.

To further customize the PDF output obtained from the Antenna House processor:

- **Edit** the transformation scenario.
- Open the **Parameters tab**.
- Add the env.AXF_OPT parameter and point to Antenna House configuration file.

The Parameters Tab

The **Parameterstab** allows you to configure the parameters sent to the DITA-OT build file.

The table displays all the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: XHTML or PDF), along with their description and current values. You can find more information about each parameter in the [DITA OT Documentation](#). You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.



Note: To input parameter values at runtime, use the *ask editor variable* in the **Value** column.

The following actions are available for managing parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Filters Tab

The **Filters** tab allows you to add filters to remove certain content elements from the generated output.

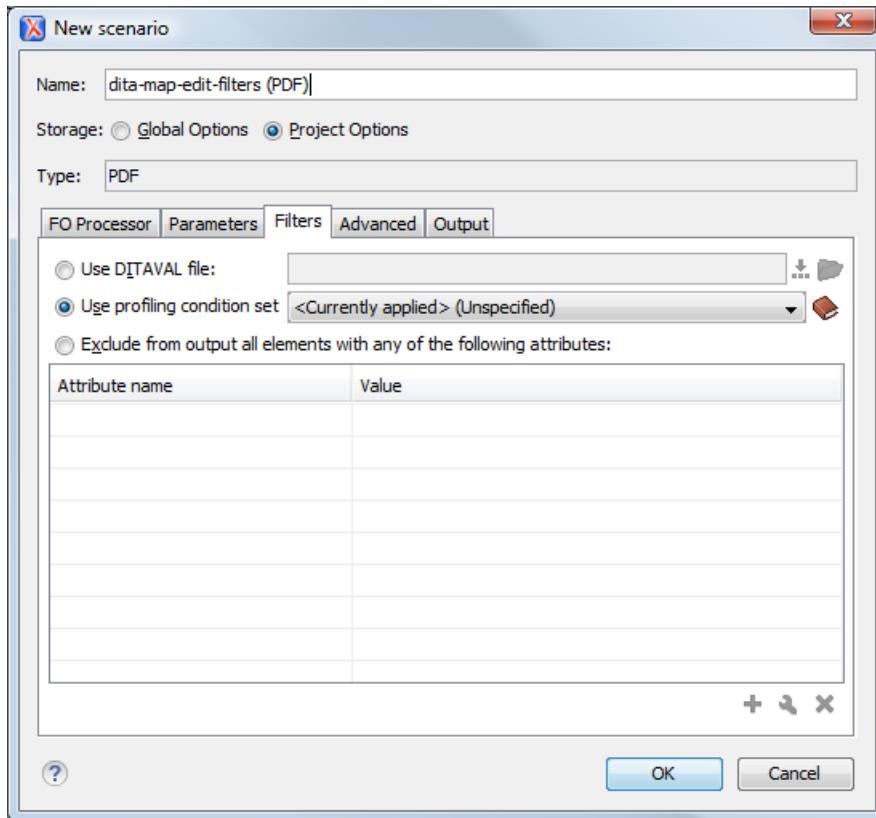


Figure 231: Edit Filters Tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the DITA map, you can specify the file to be used when filtering content. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button. You can find out more about constructing a DITAVAL file in the [DITA OT Documentation](#).
- **Use profiling condition set** - Sets the *profiling condition set* that will apply to your transformation.
- **Exclude from output all elements with any of the following attributes** - By using the **New**, **Edit**, or **Delete** buttons at the bottom of the pane, you can configure a list of attributes (name and value) to exclude all elements that contain any of these attributes from the output.

The Advanced Tab

The **Advanced** tab allows you to specify advanced options for the transformation scenario.

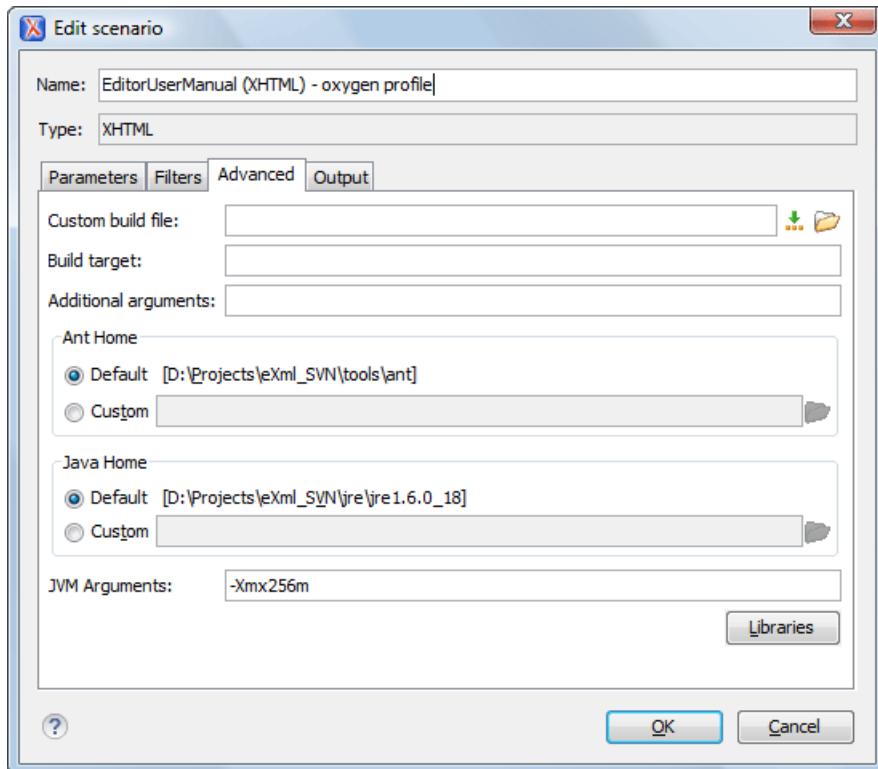


Figure 232: Advanced Settings Tab

You can specify the following parameters:

- **Custom build file** - If you use a custom DITA-OT build file, you can specify the path to the customized build file. If empty, the `build.xml` file from the `dita.dir` parameter that is configured in the **Parameters** tab is used. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button.
- **Build target** - Optionally, you can specify a build target for the build file. If no target is specified, the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation (such as `-verbose`).
- **Ant Home** - You can choose between the default or custom ANT installation to run the transformation. The default path can be configured in the [Ant preferences page](#).
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Editor.
- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by ANT. For example, if it is set to `-Xmx384m`, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (**OutOfMemoryError**).
- **Libraries** - By default, Oxygen XML Editor adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the ANT transformer.

The Output Tab

The **Output** tab allows you to configure options that are related to the location where the output is generated.

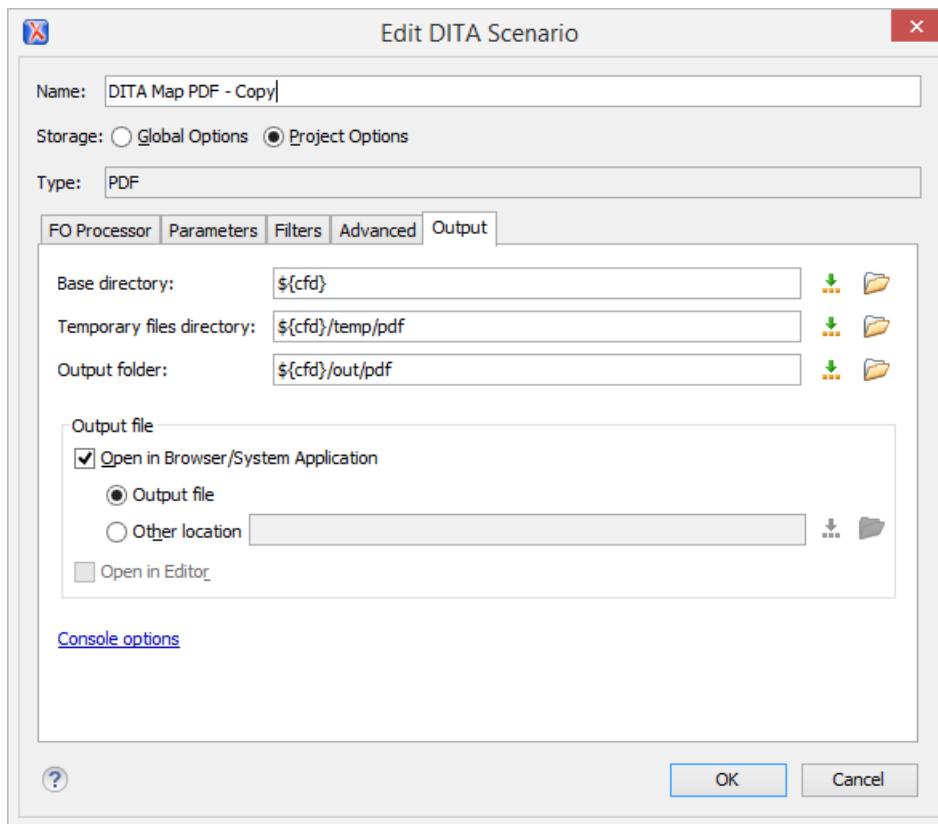


Figure 233: Output Settings Tab

You can specify the following parameters:

- **Base directory** - All the relative paths that appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Output folder** - The folder where the content of the final output is stored. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.



Note: If the DITA map or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

- **Open in Browser/System Application** - If enabled, Oxygen XML Editor automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the *Acrobat Reader* application).



Note: To set the web browser that is used for displaying HTML/XHTML pages, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor opens the file specified here. The file path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

At the bottom of the pane there is a link to the [Console options](#) preferences page that contains options to control the display of the console output received from the publishing engine.

The Skins Tab

A *skin* is a collection of CSS properties that can alter the look of the output by changing colors, font types, borders, margins, and paddings. This allows you to rapidly adapt the look and feel of the output for your organization.

Oxygen XML Editor provides a set of predefined *skins* for the **DITA Map WebHelp** and **DITA Map WebHelp with Feedback** transformation scenarios.

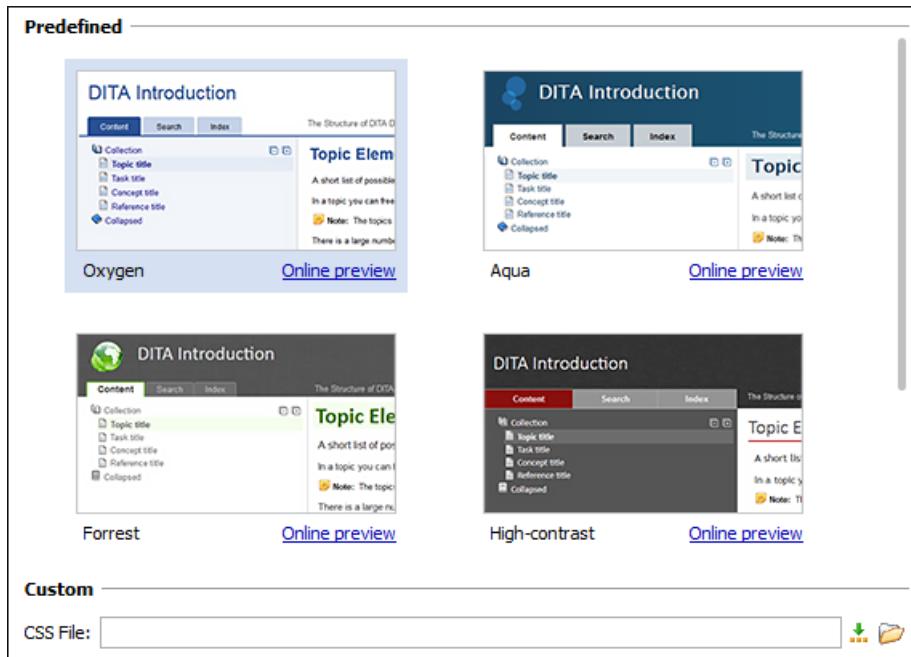


Figure 234: The Skins Tab

The predefined skins cover a wide range of chromatic themes, ranging from a very light one to a high-contrast variant. By default, the **Oxygen** skin is selected (notice the light blue border around the skin preview). If you want to obtain an output without any customization, deselect the currently selected skin.

To see how the *skin* looks when applied on a sample documentation project that is stored on the Oxygen XML Editor website, press the [Online preview](#) link.



Note: Press the [Create custom skin](#) link to open the [WebHelp Skin Builder](#) tool.

To further customize the look of the output, set the **CSS File** field to point to your custom CSS stylesheet or to a customized skin.



Note: A custom CSS file will overwrite a skin selection.



Note: The output can also be styled by setting the `args.css` parameter in the **Parameters tab**. The properties taken from the stylesheet referenced in this parameter take precedence over the properties declared in the skin set in the **Skins tab**.

DITA Map WISIWYG Transformation

Oxygen XML Editor comes bundled with a DITA OT plugin that converts a DITA Maps to PDF using a CSS based layout processor. This processor is Prince XML and is not included in the Oxygen XML Editor installation kit. It is a third-party component that needs to be purchased from <http://www.princexml.com>.

The DITA-OT plugin is located in the following directory: [oxygen Installation Directory]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.pdf.prince.

Although it includes a set of CSS files in its `css` subfolder, when this plugin is distributed in Oxygen XML Editor, the CSS files located in the `frameworks` directory takes precedence.

Creating the Transformation Scenario

To create a **DITA Map PDF WISIWYG [Experimental]** transformation scenario, follow these steps:

1. Click on the  **Configure Transformation Scenario(s)** button from the **Dita Maps Manager** toolbar.
2. Select **DITA Map PDF WISIWYG [Experimental]**.
3. When applied, this new transformation scenario uses the currently selected CSS files for the opened topic files. These CSS files can be selected from the **Styles** drop-down list from the toolbar.

 **Important:** The author could open the map in the editor and change its styles, but this is not taken into account when publishing. It seems counter intuitive, but the authors are usually editing the topics and is more probable to prefer the style used for topic editing.

4. In the **Parameters** tab, configure the following parameters:

- `prince.exec.path` - Path to the Prince executable file (for example, `c:\path\to\prince.exe` in Windows) that will be run to produce the PDF. If you installed Prince using its default settings, you can leave this blank.
- `show.changes.and.comments` - When set on `yes`, the user comments and track changes are shown in the output. Default value is `no`.

Customizing the Styles (for Output and Editing)

If you need to change the styles, make sure you install Oxygen XML Editor in a folder in which you have full read and write privileges (for instance, your user home directory). This is due to the fact that usually all the installed files are read-only (for instance, in Windows, Oxygen XML Editor is installed in the `Program Files` folder where the users do not have change rights).

If you want to change the style of an element, open a document in the editor and select **Inspect Styles** from the contextual menu. *The CSS Inspector view* will be displayed that shows all the CSS rules that apply to the selected element. Click on the link for the CSS selector that you need to change and Oxygen XML Editor will open the CSS file and position the caret at that selector. Simply add the properties you need and to see the changes in the editor, press **F5** to reload the document. Once you are satisfied with how it looks, use the transformation scenario and check for the changes in the PDF.

Set a Font for PDF Output Generated with Apache FOP

When a DITA map is transformed to PDF using the Apache FOP processor and it contains some Unicode characters that cannot be rendered by the default PDF fonts, a font that is capable of rendering these characters must be configured and embedded in the PDF result.

The settings that must be modified for configuring a font for the Apache FOP processor are detailed in [this section](#).

DITA OT PDF Font Mapping

The DITA OT contains a file

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml which maps logical fonts used in the XSLT stylesheets to physical fonts which will be used by the FO processor to generate the PDF output.

The XSLT stylesheets used to generate the XSL-FO output contain code like:

```
<xsl:attribute name="font-family">monospace</xsl:attribute>
```

The font-family is defined to be *monospace*, but *monospace* is just an alias, it is not a physical font name. So another stage in the PDF generation takes this *monospace* alias and looks in the `font-mappings.xml`.

If it finds a mapping like this:

```
<aliases>
  <alias name="monospace">Monospaced</alias>
</aliases>
```

then it looks to see if the *Monospaced* has a *logical-font* definition and if so it will use the *physical-font* specified there:

```
<logical-font name="Monospaced">
  <physical-font char-set="default">
    <font-face>Courier New, Courier</font-face>
  </physical-font>
  .....
</logical-font>
```



Important:

If no alias mapping is found for a font-family specified in the XSLT stylesheets, the processing defaults to **Helvetica**.

DITA-OT Customization

Oxygen XML Editor includes a bundled copy of the DITA-OT as an Oxygen XML Editor [framework](#). That framework includes a number of transformation scenarios for common output formats. This section explains how to customize specific parameters of a DITA transformation scenario like setting a custom DITA Open Toolkit, a custom build file or a separate installation of the Ant tool.

Support for Transformation Customizations

You can change all DITA transformation parameters to customize your needs. In addition, you can specify a custom build file, parameters to the JVM and many more for the transformation.

Using Your Custom Build File

You can specify a custom build file to be used in DITA-OT transformations by editing the transformation scenario that you are using. In the [Advanced](#) tab you should change the **Custom build file** path to point to the custom build file.

As an example, if you want to call a custom script before running the DITA OT, your custom build file would have the following content:

```
<project basedir=". " default="dist">
  <!--The DITA OT default build file-->
  <import file="build.xml"/>
  <target name="dist">
    <!-- You could run your script here -->
    <!--<exec></exec>-->
    <!--Call the DITA OT default target-->
    <antcall target="init"/>
  </target>
</project>
```

Customizing the Oxygen XML Editor Ant Tool

The Ant 1.8.2 tool which comes with Oxygen XML Editor is located in the `[OXYGEN_DIR]/tools/ant` directory. Any additional libraries for Ant must be copied to the Oxygen XML Editor Ant `lib` directory.

If you are using Java 1.6 to run Oxygen XML Editor the Ant tool should need no additional libraries to process JavaScript in build files.

Increasing the Memory for the Ant Process

For details about setting custom JVM arguments to the ANT process please see [this section](#).

Resolving Topic References Through an XML Catalog

There are situations where you want to resolve URIs with an XML catalog:

- You customized your DITA map to reference topics using URIs instead of local paths
- You have URI content references in your DITA topic files and you want to map them to local files when the map is transformed

In such situations you have to [add the catalog to Oxygen XML Editor](#). The **DITA Maps Manager** view will solve the displayed topic refs through the added XML catalog and also the DITA map transformations (for PDF output, for XHTML output, etc) will solve the URI references through the added XML catalog.

DITA to PDF Output Customization

In this topic you will see how to do a basic customization of the PDF output by setting up a customization directory.

DITA Open Toolkit PDF output customizations can be made in two major ways:

1. Creating a DITA Open Toolkit plugin which adds extensions to the PDF plugin. More details can be found in the [DITA Open Toolkit user manual](#).
2. Creating a customization directory and using it from the PDF transformation scenario. A small example of this procedure can be found below.

Let us take for example the common case of embedding a company logo image in the front matter of the book. You can later extend this example to create more complex customizations.

1. Copy the entire directory:

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/Customization to some other place, for instance: C:\Customization.

2. Copy your logo image to: C:\Customization\common\artwork\logo.png.

3. Rename C:\Customization\catalog.xml.orig to: C:\Customization\catalog.xml.

4. Open the catalog.xml in Oxygen XML Editor and uncomment this line:

```
<!--uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"-->
```

So now it looks like this:

```
<uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/>
```

5. Rename the file: C:\Customization\fo\xsl\custom.xsl.orig to: C:\Customization\fo\xsl\custom.xsl

6. Open the custom.xsl file in Oxygen XML Editor and create the template called createFrontMatter_1.0. This will override the same template from the

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/front-matter.xsl.
Now, custom.xsl has the content:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    version="1.1">

    <xsl:template name="createFrontMatter_1.0">
        <fo:page-sequence master-reference="front-matter" xsl:use-attribute-sets="__force_page_count">
            <xsl:call-template name="insertFrontMatterStaticContents"/>
            <fo:flow flow-name="xsl-region-body">
                <fo:block xsl:use-attribute-sets="__frontmatter">
                    <!-- set the title -->
                    <fo:block xsl:use-attribute-sets="__frontmatter_title">
                        <xsl:choose>
                            <xsl:when test="$map/*[contains(@class, 'topic/title')][1]">
                                <xsl:apply-templates select="$map/*[contains(@class, 'topic/title')][1]"/>
                            </xsl:when>
                            <xsl:when test="$map/*[contains(@class, 'bookmap/mainbooktitle')][1]">
                                <xsl:apply-templates select="$map/*[contains(@class, 'bookmap/mainbooktitle')][1]"/>
                            </xsl:when>
                            <xsl:when test="//*[contains(@class, 'map/map')]/@title">
                                <xsl:value-of select="//*[contains(@class, 'map/map')]/@title"/>
                            </xsl:when>
                        </xsl:choose>
                </fo:block>
            </fo:flow>
        </fo:page-sequence>
    </xsl:template>

```

```

<xsl:otherwise>
    <xsl:value-of select="/descendant::*[contains(@class, ' topic/topic
'))][1]/*[contains(@class, ' topic/title ')]"/>
        </xsl:otherwise>
    </xsl:choose>
</fo:block>

<!-- set the subtitle -->
<xsl:apply-templates select="$map//*[contains(@class, ' bookmap/booktitlealt ')]"/>

<fo:block xsl:use-attribute-sets="__frontmatter__owner">
    <xsl:apply-templates select="$map//*[contains(@class, ' bookmap/bookmeta ')]"/>
</fo:block>

<fo:block text-align="center" width="100%">
    <fo:external-graphic src="url({concat($artworkPrefix,
' /Customization/OpenTopic/common/artwork/logo.png')})"/>
</fo:block>

</fo:block>

<!--<xsl:call-template name="createPreface"/>-->

</fo:flow>
</fo:page-sequence>
<xsl:call-template name="createNotices"/>
</xsl:template>
</xsl:stylesheet>

```

7. Edit (or duplicate, then edit) the DITA Map to PDF transformation scenario. In the Parameters tab, set the customization.dir parameter to C:\Customization.

There are other ways in which you could directly modify the XSL stylesheets from the DITA OT but this customization gives you flexibility to future DITA OT upgrades in Oxygen XML Editor.

Header and Footer Customization

The XSLT stylesheet

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/static-content.xsl contains templates which output the static header and footers for various parts of the PDF like the prolog, table of contents, front matter or body.

The templates for generating a footer for pages in the body are called `insertBodyOddFooter` or `insertBodyEvenFooter`.

These templates get the static content from resource files which depend on the language used for generating the PDF. The default resource file is

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/common/vars/en.xml. These resource files contain variables like `Body odd footer` which can be set to specific user values.

Instead of modifying these resource files directly, they can be overwritten with modified versions of the resources in a PDF customization directory as explained in [DITA to PDF Output Customization](#) on page 454.

Customizing <note> Images in PDF

Here are some steps to customize the images which appear next to each type of note in the PDF output using a [PDF customization folder](#):

1. Copy the

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/common/vars/en.xml file to the [CUSTOMIZATION_DIR]\common\vars folder.

2. Edit the copied en.xml file and modify, for example, the path to the image for <note> element with the type attribute set to notice from:

```
<variable id="notice Note Image Path">Configuration/OpenTopic/cfg/common/artwork/important.png</variable>
```

to:

```
<variable id="notice Note Image Path">Customization/OpenTopic/common/artwork/notice.gif</variable>
```

3. Add your custom **notice** image to `Customization_DIR_NAME\common\artwork\notice.gif`.
4. Edit the **DITA to PDF** transformation scenario and in the **Parameters** tab set the path for the `customization.dir` property to point to the customization folder.

Creating a DITA OT Customization Plugin

To describe the steps involved in creating a **DITA Open Toolkit** plugin this section uses an example of creating an **XSLT** customization plugin that provides syntax highlighting when publishing DITA `codeblock` elements to **HTML** and **PDF** output formats. This plugin (`com.oxygenxml.highlight`) is available in the **DITA Open Toolkit** distribution that comes bundled with the latest version of Oxygen XML Editor, but these instructions show you how to create it as if it were not included.

The steps to help you to create the plugin are as follows:

1. Create a folder for your plugin in the **DITA OT plugins** folder
([`OXYGEN_DIR`]/frameworks/dita/DITA-OT/plugins/).

For example:

```
[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight
```

2. Create a **plugin.xml** file (in the same plugin folder) that contains the extension points of the plugin.



Note: You can easily create this file by using the **DITA OT Plugin** new file template that is included in Oxygen XML Editor. From the **New** file wizard you can find this template in **Framework templates > DITA > plugin**.

For example, our syntax highlighting plugin example contains the following:

```
<plugin id="com.oxygenxml.highlight">
  <feature extension="package.support.name" value="Oxygen XML Editor Support"/>
  <feature extension="package.support.email" value="support@oxygenxml.com"/>
  <feature extension="package.version" value="1.0.0"/>
  <feature extension="dita.xsl.xhtml" value="xhtmlHighlight.xsl" type="file"/>
  <feature extension="dita.xsl.xslfo" value="pdfHighlight.xsl" type="file"/>
</plugin>
```

The most important extensions in it are the references to the XSLT stylesheets that will be used to style the HTML and PDF outputs.

You can find other **DITA OT** plugin extension points here:

http://dita-ot.sourceforge.net/1.5.3/dev_ref/extension-points.html

3. Create an XSLT stylesheet to customize the output types. In our example, to customize the HTML output we need to create an XSLT stylesheet called **xhtmlHighlight.xsl** (in the same plugin folder).



Tip: You can use the **Find/Replace in Files** to find an XSLT stylesheet with content that is similar to the desired output and use it as a template to overwrite parts of your stylesheet. In our example we want to overwrite the creation of the HTML content from a DITA `codeblock` element. Since a DITA `codeblock` element has the `class` attribute value + `topic/pre-pr-d/codeblock` we can take part of the `class` attribute value (`topic/pre`) and search the **DITA OT** resources for a similar stylesheet.

Our search found the XSLT stylesheet

[`OXYGEN_DIR`]/frameworks/dita/DITA-OT/org.dita.xhtml/xsl/xslhtml/dita2htmlImpl.xsl
which contains:

```
<xsl:template match="*[contains(@class, ' topic/pre ')]" name="topic.pre">
  <xsl:apply-templates select=". " mode="pre-fmt" />
</xsl:template>
```

We use it to overwrite our **xhtmlHighlight.xsl** stylesheet, which results in the following:

```
<xsl:template match="*[contains(@class, ' topic/pre ')]" name="topic.pre">
  <!-- This template is deprecated in DITA-OT 1.7. Processing will moved into the main element rule. -->
  <xsl:if test="contains(@frame, 'top')"><hr /></xsl:if>
  <xsl:apply-templates select="*[contains(@class, ' ditao-d/ditaval-startprop ')]" mode="out-of-line"/>
  <xsl:call-template name="spec-title-nospace"/>
  <pre>
    <xsl:attribute name="class"><xsl:value-of select="name()" /></xsl:attribute>
    <xsl:call-template name="commonattributes"/>
    <xsl:call-template name="setscale"/>
    <xsl:call-template name="setidaname" />
```

```
<!--Here I'm calling the styler of the content inside the codeblock.-->
<xsl:call-template name="outputStyling"/>
</pre>
<xsl:apply-templates select="*[contains(@class, ' ditaot-d/ditaval-endprop ')]" mode="out-of-line"/>
<xsl:if test="contains(@frame, 'bot')"><hr /></xsl:if><xsl:value-of select="$newline"/>
</xsl:template>
```

You could also use another XSLT template that applies the XSLTHL library as a Java extension to style the content.

4. Create additional XSLT stylesheets to customize all other desired output types. In our example, to customize the PDF output we need to create an XSLT stylesheet called **pdfHighlight.xsl** (in the same plugin folder).

In this case we found an appropriate XSLT stylesheet

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/legacypdf/xslfo/dita2fo-elems.xsl to use as a template that we use to overwrite our **pdfHighlight.xsl** stylesheet, which results in the following:

```
<xsl:template match="*[contains(@class, ' topic/pre ')]">
  <xsl:call-template name="gen-att-label"/>
  <fo:block xsl:use-attribute-sets="pre">
    <!-- setclass -->
    <!-- set id -->
    <xsl:call-template name="setscale"/>
    <xsl:call-template name="setframe"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

 **Note:** You can edit the newly created stylesheets to customize different outputs in a variety of ways. For example, in our case you could edit the **xhtmlHighlight.xsl** or **pdfHighlight.xsl** stylesheets that we created to customize various colors for syntax highlighting.

5. To install the created plugin in the **DITA OT**, run the predefined transformation scenario called **Run DITA OT Integrator** by executing it from the  **Apply Transformation Scenario(s)** dialog. If the integrator is not visible, enable the **Show all scenarios** action that is available in the  settings drop-down list. For more information, see *Installing a Plugin in the DITA Open Toolkit* on page 457.

Results of running the integrator using our example:

XSLT content is applied with priority when publishing to both HTML and PDF outputs.

- For the HTML output, in the XSLT stylesheet

[OXYGEN_DIR]/frameworks/dita/DITA-OT/xsl/dita2html-base.xsl a new import automatically appeared:

```
<xsl:import href="../../com.oxygenxml.highlight/xhtmlHighlight.xsl"/>
```

This import is placed after all base imports and thus has a higher priority. See more about imported template precedence in the XSLT specs: <http://www.w3.org/TR/xslt#import>

- Likewise, for the PDF output, in the top-level stylesheet

[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl a new import statement appeared:

```
<xsl:import href="../../../../com.oxygenxml.highlight/pdfHighlight.xsl"/>
```

Now, you can distribute your plugin folder to anyone that has a DITA OT installation along with some simple installation notes. Your customization will work as long as the templates you are overwriting have not changed from one DITA OT distribution to the other.

Installing a Plugin in the DITA Open Toolkit

The architecture of the **DITA Open Toolkit** allows additional plugins to be installed.

1. The additional plugin(s) should be copied to the **plugins** directory of the **DITA Open Toolkit** installation (by default [OXYGEN_DIR]\frameworks\dita\DIITA-OT\plugins).

2. Run the predefined transformation scenario called **Run DITA OT Integrator** by executing it from the  **Apply Transformation Scenario(s)** dialog box. If the integrator is not visible, enable the **Show all scenarios** action that is available in the  settings drop-down list.

 **Important:** The folder where the **DITA OT** is located needs to have full write access permissions set to it.

Starting with version 17.0, Oxygen XML Editor detects the transformation type (`transtype`) declarations from **DITA OT** plugins and presents descriptions, which are contributed in the `transtype` declarations, in the **DITA Transformation Type** dialog box. Oxygen XML Editor also shows the contributed parameters from **DITA OT** plugins in the **Parameters** tab in the **Edit DITA Scenario** dialog box.

3. If the plugin contributed a new transformation type that is not detected (for instance, if you are using a previous version of Oxygen XML Editor that does not detect the `transtype` declarations), you can create a new **DITA OT** transformation scenario with a predefined type that is similar to the new transformation type. Then edit the transformation scenario, and in the **Parameters** tab add a `transtype` parameter with the value of the new transformation type.

 **Note:** A transformation type can also extend another `transtype`. For example, the `pdf-prince` `transtype` extends a `commons` transformation type that contains all the common DITA OT parameters.

Example:

```
<plugin id="com.oxygenxml.pdf.prince">
  <!-- extensions -->
  <feature extension="dita.conductor.transtype.check" value="pdf-prince" type="txt"/>
  <feature extension="dita.conductor.target.relative" value="integrator.xml" type="file"/>
  <feature extension="dita.transtype.print" value="pdf-prince"/>
  <transtype name="pdf-prince" extends="commons" desc="PDF (Prince XML - Experimental)">
    <param name="princeExecPath" type="file" desc="Path to the Prince executable file (eg:
      &quot;c:\path\to\prince.exe&quot; on Windows) which should be run to produce the PDF"/>
  </ Transtype>
</plugin>
```

DITA Specialization Support

This section explains how you can integrate and edit a DITA specialization in Oxygen XML Editor.

Integration of a DITA Specialization

A DITA specialization usually includes:

- DTD definitions for new elements as extensions of existing DITA elements
- optionally specialized processing, that is new XSLT template rules that match the extension part of the `class` attribute values of the new elements and thus extend the default processing available in DITA Open Toolkit

A specialization can be integrated in the application with minimum effort:

1. If the DITA specialization is available as a DITA Open Toolkit plugin, copy the plugin to the location of the DITA OT you are using (by default `[OXYGEN_DIR]\frameworks\dita\{DITA-OT}\plugins`). Then run the DITA OT integrator to integrate the plugin. In the **Transformation Scenarios view** there is a predefined scenario called **Run DITA OT Integrator** which can be used for this.

 **Important:** The directory where the DITA OT is located needs to have full write access permissions set to it.

2. If the specialization is not available as a plugin, you have the following options:

- If the DTD's that define the extension elements are located in a folder outside the DITA Open Toolkit folder, add new rules to the DITA OT catalog file for resolving the DTD references from the DITA files that use the specialized elements to that folder. This allows correct resolution of DTD references to your local DTD files and is needed for both validation and transformation of the DITA maps or topics. The DITA OT catalog file is called `catalog-dita.xml` and is located in the root folder of the DITA Open Toolkit.

- If there is specialized processing provided by XSLT stylesheets that override the default stylesheets from DITA OT, these new stylesheets must be called from the Ant build scripts of DITA OT.
-  **Important:** If you are using DITA specialization elements in your DITA files, it is recommended that you activate the **Enable DTD/XML Schema processing in document type detection** checkbox in the [Document Type Association page](#).

Editing DITA Map Specializations

In addition to recognizing the default DITA map formats: `map` and `bookmap` the **DITA Maps Manager** view can also be used to open and edit specializations of DITA Maps.

All advanced edit actions available for the map like insertion of topic refs, heads, properties editing, allow the user to specify the element in an editable combo box. Moreover the elements which appear initially in the combo are all the elements which are allowed to appear at the insert position for the given specialization.

The topic titles rendered in the **DITA Maps Manager** view are collected from the target files by matching the `class` attribute and not a specific element name.

When editing DITA specializations of maps in the main editor the insertions of topic reference, topic heading, topic group and conref actions should work without modification. For the table actions you have to modify each action by hand to insert the correct element name at caret position. You can go to the **DITA Map** document type from the [Document Type Association page](#) and edit the table actions to insert the element names as specified in your specialization. See [this section](#) for more details.

Editing DITA Topic Specializations

In addition to recognizing the default DITA topic formats: `topic`, `task`, `concept`, `reference` and `composite`, topic specializations can also be edited in the **Author** mode.

The content completion should work without additional modifications and you can choose the tags that are allowed at the caret position.

The CSS styles in which the elements are rendered should also work on the specialized topics without additional modifications.

The toolbar/menu actions should be customized to insert the correct element names. You can go to the DITA document type from the [Document Type Association page](#) and edit the actions to insert the element names, as specified in your specialization. See [this section](#) for more details.

Use an External DITA Open Toolkit in Oxygen XML Editor

Oxygen XML Editor comes bundled with a DITA Open Toolkit, located in the `[OXYGEN_DIR]/frameworks/dita/DITA-OT` directory. Starting with Oxygen XML Editor version 17, if you want to use the external DITA OT for all transformations and validations, you can [open the Preferences dialog box](#) and go to [the DITA page](#), where you can specify the DITA OT to be used. Otherwise, to use an external DITA Open Toolkit, follow these steps:

1. Edit your transformation scenarios and in the **Parameters** tab change the value for the `dita.dir` parameter to point to the new directory.
2. To make changes in the libraries that come with the DITA Open Toolkit and are used by the ANT process, go to the **Advanced** tab, click the **Libraries** button and uncheck **Allow Oxygen to add high priority libraries to classpath**.
3. If there are also changes in the DTDs and you want to use the new versions for content completion and validation, go to the Oxygen XML Editor preferences in the **Document Type Association** page, edit the **DITA** and **DITA Map** document types and modify the catalog entry in the **Catalogs** tab to point to the custom catalog file `catalog-dita.xml`.

Reusing Content

DITA allows you to reuse content from other DITA files with a content reference in the following ways:

- You can select content in a topic, create a reusable component from it and reference the component in other locations using the actions **Create Reusable Component** and **Insert Reusable Component**. A reusable component is a file, usually shorter than a topic. You also have the option of replacing the selection with the component that you are in the process of creating. The created reusable component file is usually self-contained and it's automatically generated content can be fine tuned by modifying the resources located in the folder
[OXYGEN_DIR]\frameworks\dita\reuse.
- You can add, edit, and remove a content reference (`conref`) attribute to/from an existing element. The actions **Add/Edit Content Reference** and **Remove Content Reference** are available on the contextual menu of the Author editor and on the DITA menu. When a content reference is added or an existing content reference is edited, you can select any topic ID or interval of topic IDs (set also the `conrefend` field in the dialog box for adding/editing the content reference) from a target DITA topic file.
- You can insert an element with a content reference (`conref` or `conkeyref`) attribute using one of the actions **Insert Content Reference** and **Insert Content Key Reference** that are available on the DITA menu, the Author custom actions toolbar and the contextual menu of the Author editor.

DITA makes the distinction between local content, that is the text and graphics that are actually present in the element, and referenced content that is referenced by the element but is located in a different file. To display referenced content, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Display referenced content**.

Working with Content References

The DITA `conref` feature (short for *content reference*) lets you include a piece of source content by reference in other topics. When you need to update that content, you do it in only one place. Typical uses of content references are for product names, warnings, definitions, or process steps.

You can use either or both of the following strategies for managing content references:

- *Reusable components* - With this strategy, you create a new file for each piece of content that you want to reuse.
- *Arbitrary content references* - You may prefer to keep many pieces of reusable content in one file. For example, you might want one file to consist of a list of product names, with each product name in a phrase (`<ph>` element) within the file. Then, wherever you need to display a product name, you can insert a content reference that points to the appropriate `<ph>` element in this file.



Note: A reference displays tracked changes and also comments of the source fragment. To edit these comments or accept/reject the changes, right click them and select **Edit Reference**.

This strategy requires more setup than reusable components, but makes easier centrally managing the reused content.

Oxygen XML Editor creates a reference to the external content by adding a `conref` attribute to an element in the local document. The `conref` attribute defines a link to the referenced content, made up of a path to the file and the topic ID within the file. The path may also reference a specific element ID within the topic. Referenced content is not physically copied to the referencing file, but Oxygen XML Editor displays it as if it is there in the referencing file. You can also choose to view local content instead of referenced content, to edit the attributes or contents of the referencing element.



Note: To search for references made through a direct content reference of a topic, paragraph, list item, and so on, use the **Search References** action from the contextual menu.

How to Work with Reusable Components

When you need to reuse a part of a DITA topic in different places (in the same topic or in different topics) it is recommended to create a separate component and insert only a reference to the new component in all places. Below are the steps for extracting a reusable component, inserting a reference to the component and quickly editing the content inside the component.

1. Select with the mouse the content that you want to reuse in the DITA file opened in **Author** mode.
2. Start the action **Create Reusable Component** that is available on the DITA menu, the Author framework actions toolbar and the contextual menu of the Author editor.
3. In the combo box **Reuse Content** select the DITA element with the content that you want to extract in a separate component. The combo box contains the current DITA element where the cursor is located (for example a **p** element - a paragraph - or a **step** or a **taskbody** or a **conbody** etc.) and also all the ancestor elements of the current element.
4. In the **Description** area enter a textual description for quick identification by other users of the component.
5. If you want to replace the extracted content with a reference to the new component you should leave the checkbox **Replace selection with content reference** with the default value (selected).
6. Press the **Save** button, which will open a file system dialog box where you have to select the folder and enter the name of the file that will store the reusable component.
7. Press the **Save** button in the file system dialog box to save the reusable component in a file. If the checkbox was selected in the **Create Reusable Component** dialog box, the **conref** attribute will be added to the element that was extracted as a separate component. In **Author** mode the content that is referenced by the **conref** attribute is displayed with grey background and is read-only because it is stored in other file.
8. Optionally, to insert a reference to the same component in other location just place the cursor at the insert location and run the action **Insert Reusable Component** that is available on the DITA menu, the Author framework actions toolbar and the contextual menu of the Author editor. In the file system dialog box, Just select the file that stores the component and press the **OK** button. The action will add a **conref** attribute to the DITA element at the insert location. The referenced content will be displayed in **Author** mode with grey background to indicate that it is not editable.
9. Optionally, to edit the content inside the component just click on the  **Edit Content** icon at the start of the grey background area which will open the component in a separate editor.

Insert a Direct Content Reference

You can use the same content in multiple topics by inserting a DITA content reference to that content. The following steps describe the procedure of inserting a DITA content reference:

1. Position your caret inside the element that you want to reference and in the **Attributes view** enter a value in the **ID** field.
In case you want to reuse just a part of the content of an element, select the content with your cursor, press **Enter** and in the proposals list select **ph**. This encapsulates your content inside a **phrase** (**<ph>**) element, allowing you to set an ID and then reference it.
2. Open the topic where you want to insert the reference to this element.
3. Click  **Insert a DITA Content Reference** on the main toolbar.
The **Insert Content Reference** dialog box is displayed.
4. In the **Insert Content Reference** dialog box, from the **URL** field, navigate to the topic that holds the element you want to reference.
In the **Target ID** section of the **Insert Content Reference** dialog box, Oxygen XML Editor presents the elements that you can reference.
5. Click the ID of the element you want to reference, then click **OK**.
In case you select an interval of elements, the **Conrefend** field is filled with the **id** value of the element that ends the selected interval.
A reference to the selected element is inserted at the caret position.

The Insert Content Reference Dialog Box

The **Insert Content Reference** dialog box lets you reuse content by inserting references to the DITA elements that hold the content you want to reuse.



Note: To reference the content inside a DITA element you first have to set an ID for that element.

The DITA `conref` attribute provides a mechanism for reuse of content fragments. The `conref` attribute stores a reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element. For more details, go to <http://docs.oasis-open.org/dita/v1.0/archspec/conref.html>.

Oxygen XML Editor *displays the referenced content* of a DITA `conref` if it can resolve it to a valid resource. If you have URI's instead of local paths in the XML documents and your DITA OT transformation needs an XML catalog to map the URI's to local paths you have to [add the catalog to Oxygen XML Editor](#). If the URI's can be resolved, the referenced content is displayed in the **Author** mode and in the transformation output.

To open the **Insert Content Reference** dialog box, do one of the following:

- Go to **DITA** >  **Insert a DITA Content Reference**.
- Click the  **Insert a DITA Content Reference** action on the main toolbar.
- In the contextual menu of the editing area, go to **Reuse** >  **Insert a DITA Content Reference**.

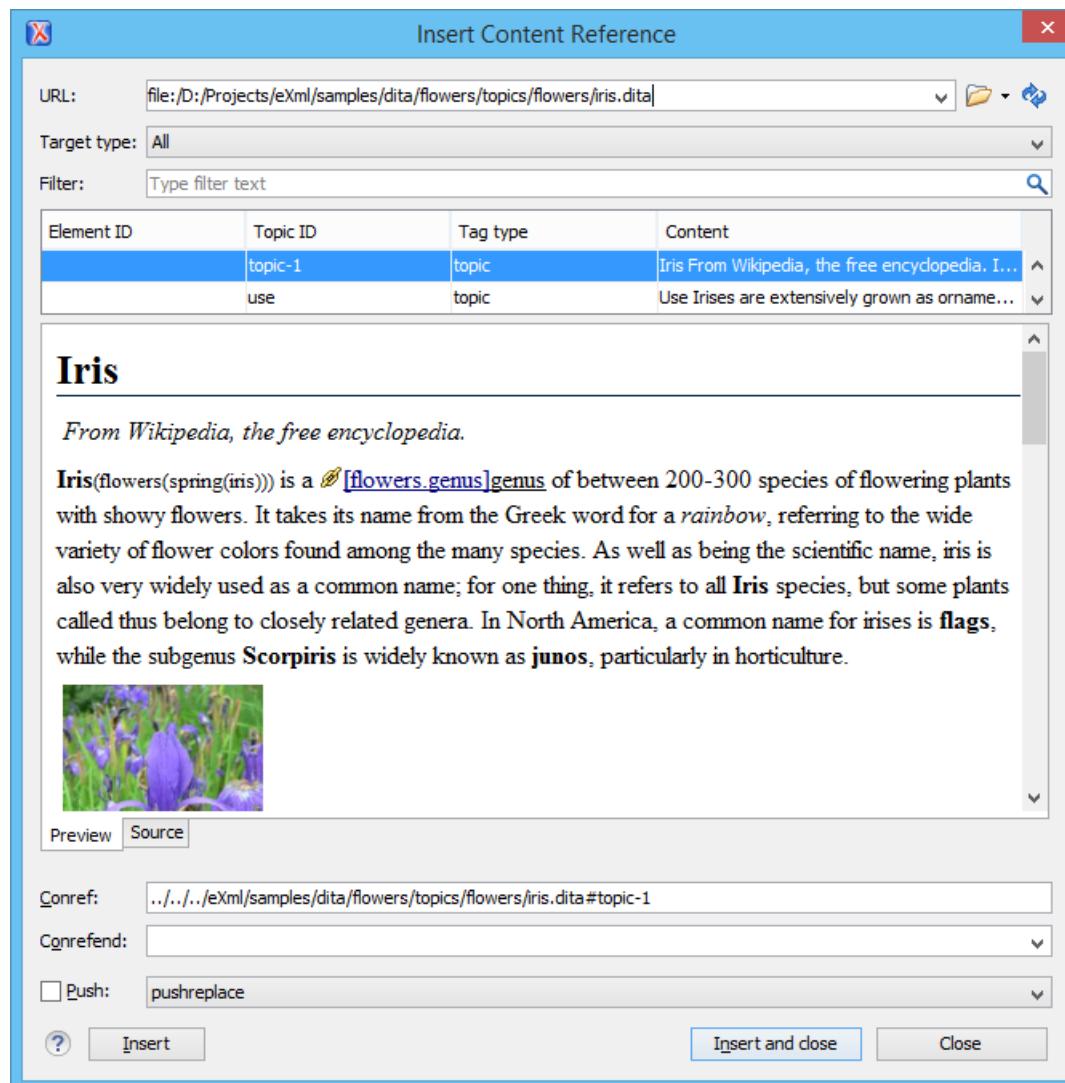


Figure 235: The Insert Content Reference Dialog Box



Note: The **Insert Content Reference** dialog box is not modal. The dialog box is closed automatically if you switch to a different editor.

The following fields are available in this dialog box:

- **URL** - specifies the path to the topic that holds the content you want to reference.
- **Target type** - specifies the type of the element to which you are targeting your `conref`.
- **Target ID** - presents all the element IDs defined in the source topic.
- **Preview** - displays a preview of the content in the element that you select in the **Target ID** list.
- **Source** - displays the source of the element your want to reference.
- **Conref** - displays the value of the `conref` attribute.
- **Conrefend** - in case you select an interval of elements, this field displays the end value of the `conref` attribute.
- **Push** - this option enables you to push content into DITA topics and maps, provided that the topics and maps contain elements with `id` attributes that identify the places where the content is to be pushed.

Moving and Renaming Resources

You can move or rename resources on disk directly from Oxygen XML Editor. To do this, use one of the following actions available in the contextual menu of the **DITA Maps Manager** view:

Rename resource

This action allows you to change the name of a resource linked in the edited DITA Map, using the **Rename resource** dialog box. This dialog box contains the following options:

- **Update references** - Enable this checkbox to update all references of the file in the edited DITA Map and in the files referenced from the DITA Map. This way, the completeness of the DITA Map is preserved.
- **Preview** - Select this button to display a preview of the changes Oxygen XML Editor is about to make.
- **Rename** - Executes the **Rename resource** operation.
- **Cancel** - Cancels the **Rename resource** operation. No changes are applied.

Move resource

This action allows you to change the location of a resource linked in the edited DITA Map, using the **Move resource** dialog box. This dialog box contains the following options:

- **Destination** - Specifies the target location on disk of the edited resource.
- **File name** - Allows you to change the name of the edited resource.
- **Update references** - Enable this checkbox to update all references of the file in the edited DITA Map and in the files referenced from the DITA Map. This way, the completeness of the DITA Map is preserved.
- **Preview** - Select this button to display a preview of the changes Oxygen XML Editor is about to make.
- **Move** - Moves the edited resource in the target location on disk.
- **Cancel** - Cancels the **Move resource** operation. No changes are applied.



Note: If a root DITA Map is not defined, the move and rename actions are executed in the context of the current DITA Map.

DITA Profiling / Conditional Text

Conditional text is a way to mark blocks of text meant to appear in some renditions of the document, but not in others. It differs from one variant of the document to another, while unconditional text appear in all document versions.

For instance you can mark a section of a document to be included in the manual designated for the *expert* users, other for the *novice* users manual while unmarked sections are included in any rendition.

You can use conditional text when you develop documentation for:

- A series of similar products
- Different releases of a product
- Various audiences

The benefits of using conditional text include reduced effort for updating and translating your content and an easy way to customize the output for various audiences.

Oxygen XML Editor offers full support for DITA conditional text processing: profiling attributes can be easily managed to filter content in the published output. You can toggle between different profile sets to see how the edited content looks like before publishing.

DITA offers support for profiling/conditional text by using profiling attributes. With Oxygen XML Editor you can define values for the DITA profiling attributes. The profiling configuration can be shared between content authors through the project file. There is no need for coding or editing configuration files.

Several profiling attributes can be aggregated into a profiling condition set that allow you to apply more complex filters on the document content. A Profiling Condition Set is a very powerful and convenient tool used to preview the content that goes into the published output. For example, an installation manual available both in Windows and Linux variants can be profiled to highlight only the Linux procedures for more advanced users.

To watch our video demonstration about DITA profiling, go to http://oxygentools.com/demo/DITA_Profiling.html.

Profiling / Conditional Text Markers

If the **Show Profiling Attributes** option (available in the **Tools** menu) is enabled, all profiling attributes set on the current element are listed at the end of the highlighted block. Profiled text is marked in the **Author** mode with a light green border.

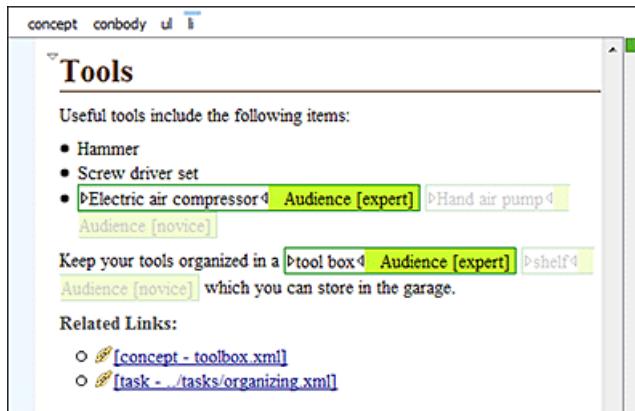


Figure 236: Profiling in Author

In the **DITA Maps Manager View**, the following icons are used to mark profiled and non-profiled topics:

- - the topic contains profiling attributes
- - the topic inherits profiling attribute from its ancestors
- - the topic contains and inherits profiling attributes
- - (dash) - the topic neither contains, nor inherits profiling attributes

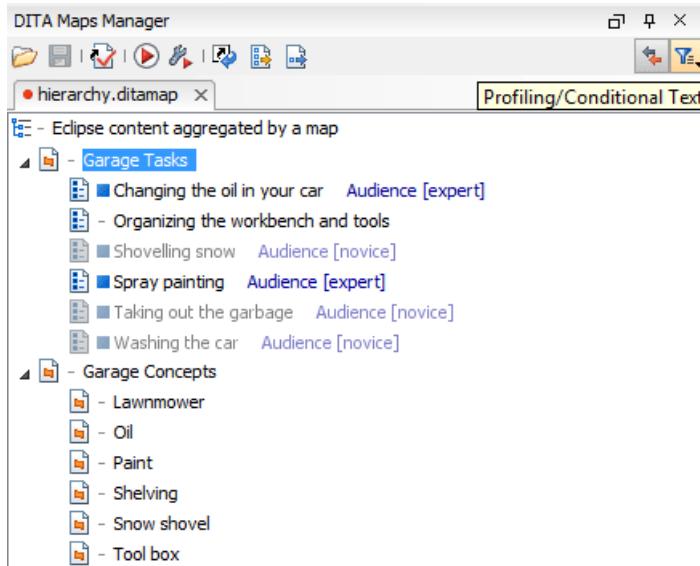


Figure 237: Profiling in DITA Maps Manager

The profiled content that does not match the rules imposed by the current condition sets is grayed-out, meaning that it will not be included in the published output.

Profiling with a Subject Scheme Map

A subject scheme map allows you to create custom profiling values and to manage the profiling attribute values used in the DITA topics without having to write a DITA specialization.

Subject scheme maps use key definitions to define a collection of profiling values. A map that uses the set of profiling values must reference at its highest level the subject scheme map in which the profiling values are defined, for example:

```
<topicref href="test.ditamap" format="ditamap" type="subjectScheme" />
```

A profiled value should be a short and readable keyword that identifies a metadata attribute. For example, the audience metadata attribute may take a value that identifies the user group associated with a particular content unit. Typical user values for a medical-equipment product line might include `therapist`, `oncologist`, `physicist`, `radiologist`, `surgeon`, and so on. A subject scheme map can define a list of these audience values.

The following is an example of content from a subject scheme:

```
<subjectScheme>
  <!-- Pull in a scheme that defines audience user values -->
  <subjectdef keys="users">
    <subjectdef keys="therapist"/>
    <subjectdef keys="oncologist"/>
    <subjectdef keys="physicist"/>
    <subjectdef keys="radiologist"/>
    <subjectdef keys="surgeon">
      <subjectdef keys="neuro-surgeon">
        <subjectdef keys="plastic-surgeon"/>
      </subjectdef>
    </subjectdef>
  </subjectdef>
  <!-- Define an enumeration of the audience attribute, equal to
      each value in the users subject. This makes the following values
      valid for the audience attribute: therapist, oncologist, physicist, radiologist -->
  <enumerationdef>
    <attributedef name="audience"/>
    <subjectdef keyref="users"/>
  </enumerationdef>
</subjectScheme>
```

When you edit a DITA topic in the **Text** or **Author** mode, Oxygen XML Editor collects all the profiling values from the Subject Scheme Map that is referenced in the map that is currently opened in the *DITA Maps Manager*. The values of profiling attribute defined in a Subject Scheme Map are available in the **Edit Profiling Attribute** dialog regardless of their mapping the **Conditional Text** preferences page.

In the example above, the values therapist, oncologist, physicist, and so on, are displayed in the *content completion window* as values for the audience attribute.

Now let us consider we have the following fragment in a topic:

```
<p audience="neuro-surgeon">Some text.. </p>
```

When you define a DITAVAL filter, you can, for instance, exclude anything that is profiled as surgeon:

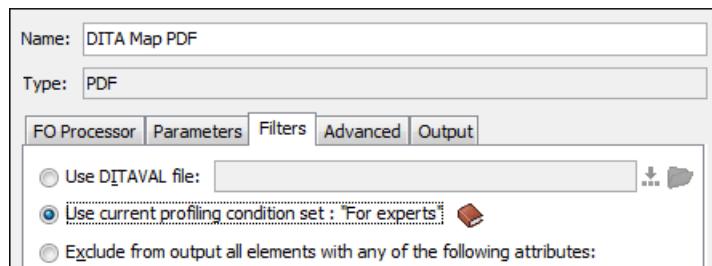
```
<val>
  <prop action="exclude" att="audience" val="surgeon"/>
</val>
```

If you then transform the main DITA Map specifying the DITAVAL filter file in the transformation scenario, the p element should be excluded from the output. Thus, excluding the surgeon audience also excludes neuro-surgeon and plastic-surgeon from the output. More details about how hierarchical filtering and Subject Scheme Maps should work are found in the following specification

<http://docs.oasis-open.org/dita/v1.2/os/specLangSubjectScheme.html#subjectScheme>

Publish Profiled Text

Oxygen XML Editor comes with preconfigured transformation scenarios for DITA. By default, these scenarios take the current profiling condition set into account during the transformation, as defined in the **Filters** tab when *creating a DITA transformation*.



How to Profile DITA Content

1. Open the *Preferences dialog box*, go to **Editor > Edit modes > Author > Profiling / Conditional Text**, and edit the **Profiling Attributes** table.

Note that this table will be ignored if a *Subject Scheme Map* is in use.

2. For DITA documents, there are already default entries for audience, platform, product, otherprops and rev. You can customize the attributes and their values.

This is a one-time operation. Once you save the customized attributes and values, you can use them to profile any DITA project.

3. To use the profiling attributes set in the previous step, do one of the following:

- a) Right-click (**Command Click on OS X**) a topic reference in **DITA Maps Manager** and choose **Edit Profiling Attributes** from the contextual menu.
- b) In the **Author** editing mode, right-click (**Command Click on OS X**) an XML element and choose **Edit Profiling Attributes** from the contextual menu.
- c) Use the **Attributes** view to set profiling attributes.

Enable the **Show Profiling Attributes** option to display the profiling markup in the **Author** editing mode.

Working with MathML

You can add MathML equations in a DITA document using one of the following methods:

- Embed MathML directly into a DITA topic. You can start with the **Framework templates / DITA / topic / Composite with MathML** document template, available from the **New** file action wizard.
- Reference an external MathML file as an image, using the **Insert Image Reference** toolbar action.



Note: MathML equations contained in DITA topics can only be published out-of-the-box in PDF using the **DITA PDF** transformation scenario. For other publishing formats, you must employ additional customizations for handling MathML content.

MathML Equations in the HTML Output

Currently, only **Firefox** can render **MathML** equations embedded in the **HTML** code. **MathJax** is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

If you have DocBook or DITA content that has embedded **MathML** equations and you want to properly view the equations in published HTML output types (WebHelp, CHM, EPUB, etc.), you need to add a reference to the MathJax script in the **head** of all HTML files that have the equation embedded.

For example:

```
<script type="text/javascript" src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

For DITA documents, you can also edit the **DITA Map WebHelp** transformation scenario and set the `args.hdf` parameter to point to the `footer.html` resource. Then transform to WebHelp and the equation should be properly rendered in the browsers such as IE, Chrome, and Firefox.

Chapter

9

Predefined Document Types

Topics:

- [Document Type](#)

The following pre-defined document types are supported in Oxygen XML Editor and each of these document types include built-in transformation scenarios, document templates , and Author extension actions:

- [DocBook 4](#) - A document type standard for books, articles, and other prose documents (particularly technical documentation).
- [DocBook 5](#) - An enhanced (version 5) document type standard designed for a variety of documents (particularly technical documentation).
- [DITA](#) - An XML-based architecture designed for authoring, producing, and delivering technical information.
- [DITA Map](#) - A document type that collects and organizes references to DITA topics or other maps.
- [XHTML](#) - Extensible HyperText Markup Language includes the same depth of expression as HTML, but also conforms to XML syntax.
- [TEI ODD](#) - Text Encoding Initiative One Document Does it all is an XML-conformant specification that allows you to create TEI P5 schema in a literate programming style.
- [TEI P4](#) - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
- [TEI P5](#) - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
- [JATS](#) - The NISO Journal Article Tag Suite is a technical standard that defines an XML format for scientific literature.

Oxygen XML Editor also provides limited support and includes document templates for a variety of other document types, including:

- [EPUB \(NCX, OCF, OPF 2.0 & 3.0\)](#) - A standard for e-book files.
- [DocBook Targetset](#) - For resolving cross-references when using *olinks*.
- [ANT Build Scripts](#) - A tool for automating software build processes, written in Java and primarily intended for use with Java.
- [XSLT Stylesheets](#) - A document type that provides a visual mode for editing XSLT stylesheets.
- [WSDL](#) - Web Services Description Language is an XML language for describing the functionality offered by a web service.
- [Schematron](#) - For making assertions about the presence or absence of patterns in XML documents. This document type applies to the ISO Schematron version.
- [Schematron Quick Fixes \(SQF\)](#) - An extension of the ISO standard Schematron, allows developers to define *QuickFixes* for Schematron errors.
- [StratML \(Part 1 & 2\)](#) - Part 1 and 2 of the Strategy Markup Language specification.
- [XProc](#) - A document type for processing XProc script files.

- [*XML Schema*](#) - Documents that provide support for editing annotations.
- [*SVG*](#) - Scalable Vector Graphics is a language for describing two-dimensional graphics in XML.
- [*MathML*](#) - Mathematical Markup Language (2.0 and 3.0) is an application of XML for describing mathematical notations.
- XML Spec - A markup language for W3C specifications and other technical reports.
- DITAVAL - DITA conditional processing profile to identify the values you want to conditionally process for a particular output, build, or other purpose.
- Daisy - A technical standard for digital audio books, periodicals, and computerized text. It is designed to be an audio substitute for print material.
- EAD - Encoded Archival Description is an XML standard for encoding archival finding aids.
- KML - Keyhole Markup Language is an XML notation for expressing geographic visualization in maps and browsers.
- Maven Project & Settings - Project or settings file for Maven build automation tool that is primarily used for Java projects.
- Oasis XML Catalog - A document that describes a mapping between external entity references and locally-cached equivalents.
- XLIFF (1.2 & 2.0) - XML Localization Interchange File Format is a standard for passing data between tools during a localization process.

Document Type

A *document type* or *framework* is associated to an XML file according to a set of rules. It also includes a variety of settings that improve editing capabilities in the **Author** mode for its particular file type. These settings include:

- A default grammar used for validation and content completion in both **Author** mode and **Text** mode.
- CSS stylesheets for rendering XML documents in **Author** mode.
- User actions invoked from toolbar or menu actions in **Author** mode.
- Predefined scenarios used for transformations for the class of XML documents defined by the document type.
- XML catalogs.
- Directories with file templates.
- User-defined extensions for customizing the interaction with the content author in **Author** mode.

Oxygen XML Editor comes with built-in support for many common document types. Each document type is defined in a framework. You can create new frameworks or make changes to existing frameworks to suit your individual requirements.

To see a video on configuring a framework in Oxygen XML Editor, go to
<http://oxygentools.com/demo/FrameworkConfiguration.html>.

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a *DocBook 4* document when one of the following conditions are true:

- The root element name is `book` or `article`.
- The PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`.

The default schema, `docbookx.dtd`, for these documents is stored in
`[OXYGEN_DIR]/frameworks/docbook/4.5/dtd/`.

The default CSS files used for rendering DocBook content in **Author** mode are stored in
`[OXYGEN_DIR]/frameworks/docbook/css/`.

The default XML catalog, `catalog.xml`, is stored in `[OXYGEN_DIR]/frameworks/docbook/`.

To watch our video demonstration about editing DocBook documents, go to
http://oxygentools.com/demo/DocBook_Editing_in_Author.html.

DocBook 4 Author Actions

A variety of actions are available in the DocBook 4 framework that can be added to the **DocBook4** menu, the **Author custom actions** toolbar, the contextual menu, and the **Content Completion Assistant**. The following default actions are included in the toolbar and the **DocBook4** menu and are readily available when editing in **Author** mode (most of them are also available, by default, in the contextual menu):

B Bold emphasized text

Emphasizes the selected text by surrounding it with `<emphasis role="bold">`. You can use this action on multiple non-contiguous selections.

I Italic emphasized text

Emphasizes the selected text by surrounding it with `<emphasis role="italic">`. You can use this action on multiple non-contiguous selections.

U Underline emphasized text

Emphasizes the selected text by surrounding it with `<emphasis role="underline">`. You can use this action on multiple non-contiguous selections.

Link Actions Drop-Down List

The following link actions are available from this list:

- **Cross reference (link)** - Inserts a hypertext link.
- **Cross reference (xref)** - Inserts a cross reference to another part of the document.
- **Web Link (ulink)** - Inserts a link that addresses its target with a URL (Universal Resource Locator).
- **Insert olink** - Inserts a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values that are present in a *Targetset* file.

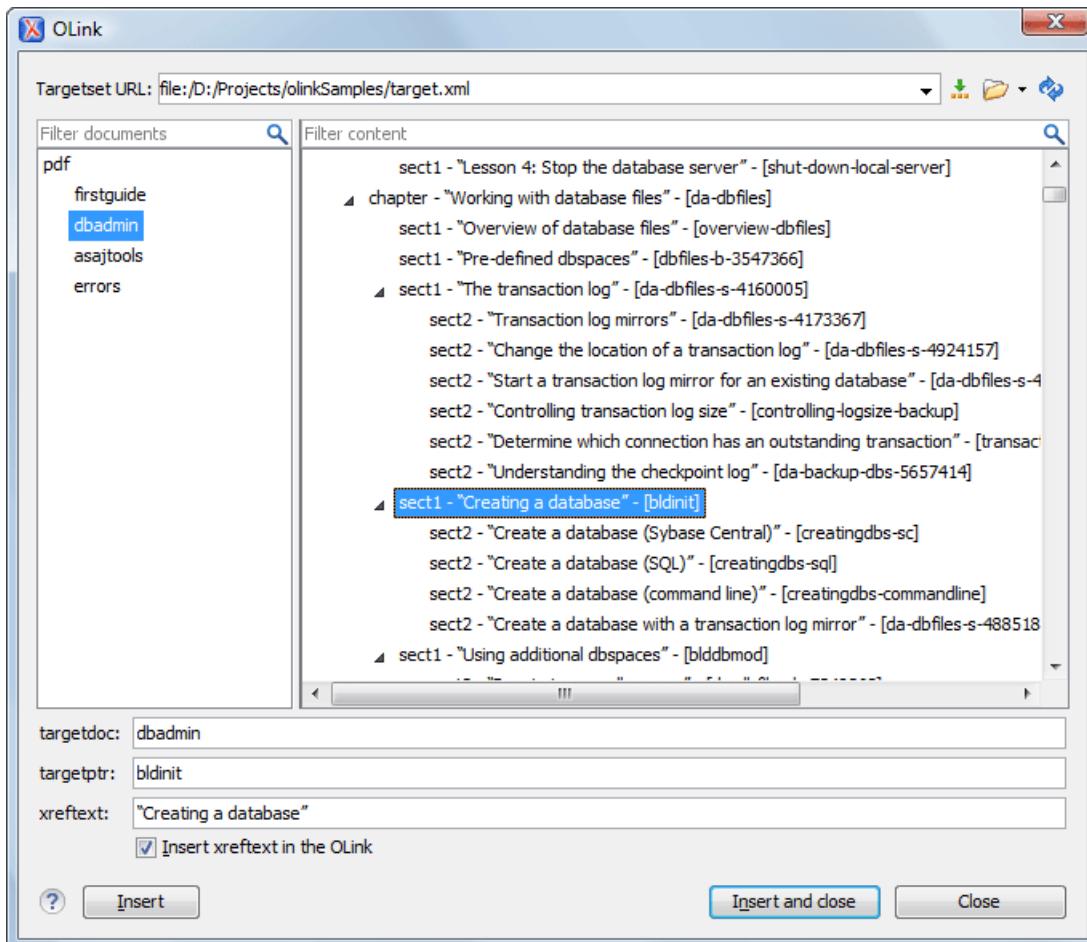


Figure 238: Insert OLink Dialog Box

After you choose the **Targetset URL**, the structure of the target documents is presented. For each target document (`targetdoc`), the content is displayed allowing you to easily identify the `targetptr` for the `olink` element that will be inserted. You can use the search fields to quickly identify a target. If you already know the values for the `targetdoc` and `targetptr`, you can insert them directly in the corresponding fields. You can also edit an `olink` using the **Edit OLink** action that is available on the contextual menu. The last used **Targetset URL** will be used to identify the edited target.

- **Insert URI** - Inserts an `URI` element. The `URI` identifies a Uniform Resource Identifier (URI) in content.

Insert image reference

Inserts an image reference at the caret position. Depending on the current context, an image-type element is inserted.

Insert XInclude

Opens a dialog box that allows you to browse and select content to be included and automatically generates the corresponding `XInclude` instruction.

§ ▾ Insert Section Drop-Down List

The following link actions are available from this list:

-  **Insert Section** - Inserts a new section or subsection in the document, depending on the current context. For example, if the current context is `sect1` then a `sect2` is inserted. By default, this action also inserts a `para` element as a child node. The `para` element can be deleted if it is not needed.
-  **Promote Section** - Inserts the current node as a brother of the parent node.
-  **Demote Section** - Inserts the current node a child of the previous node.

¶ Insert a new paragraph

Insert a new paragraph at current cursor position.

Σ Insert a MathML equation

Opens the **XML Fragment Editor** that allows you to insert and [edit MathML notations](#).

▫ Insert a step or list Item

Inserts a new step or list item in the current list type.

☰ Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also automatically inserted by default.

☰ Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also automatically inserted by default.

☰ Insert a variable list at the caret position

Inserts a DocBook variable list. A child list item is also inserted automatically by default.

☰ Insert a procedure

Inserts a DocBook `procedure` element. A step `child` item is also inserted automatically.

⤒ Sort

Sorts a table selection.

⤒ Insert Table

Opens a dialog box that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

⤒ Insert a new table row below the current row

Inserts a new table row with empty cells below the current row. This action is available when the caret is positioned inside a table.

⤒ Insert a new table column after the current column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

⤒ Insert a table cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, Oxygen XML Editor a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

⤒ Delete a table column

Deletes the table column located at caret position.

⤒ Delete a table row

Deletes the table row located at caret position.

⤒ Edit Table Properties

Opens the **Table properties** dialog box that allows you to configure properties of a table (such as frame borders).

Table Join/Split Drop-Down List

The following link actions are available from this list:

-  **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
-  **Join Cell Above** - Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.
-  **Join Cell Below** - Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.



Note: When you use  **Join Cell Above** and  **Join Cell Below**, Oxygen XML Editor deletes the source row if it remains empty. The cells that span over multiple rows are also updated.

-  **Split Cell To The Left** - Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column. Oxygen XML Editor decreases the column span of the source cell with one.
-  **Split Cell To The Right** - Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column. Oxygen XML Editor decreases the column span of the source cell with one.
-  **Split Cell Above** - Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row. Oxygen XML Editor decreases the column span of the source cell with one.
-  **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row. Oxygen XML Editor decreases the column span of the source cell with one.

The following default actions are available in the **Docbook4** menu:

ID Options

Opens the **ID Options** dialog box that allows you to specify the elements for which Oxygen XML Editor generates a unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported [Editor Variables](#) on page 557.

To retain the element IDs when copying content in a document, disable the **Remove IDs when copying content in the same document** option.

Generate IDs

This action generates and sets unique IDs for:

- The element at caret position.
- All top-level elements found in the current selection.
- Selections that contain elements from the **ID Options** list.



Note: IDs that were previously set are preserved.

Drag/Drop Actions

Dragging a file from [the Project view](#) or [DITA Maps Manager view](#) and dropping it into a DocBook 4 document that is edited in **Author** mode, creates a link to the dragged file (the `ulink` DocBook element) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into a DocBook 4 document inserts an image element (the `inlinegraphic` DocBook element with the `fileref` attribute) at the drop location, similar to the  **Insert Image Reference** toolbar action.

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, EPUB and EPUB 3.

WebHelp Output

DocBook 4 documents can be transformed into WebHelp systems, such as:

WebHelp Output

To publish DocBook 4 to WebHelp, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp** transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use_stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `webhelp.copyright` - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
    js = d.createElement(s); js.id = id; js.src =
    "/connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `l10n_gentext_default_language` - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- `webhelp.search.ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).

WebHelp With Feedback Output

To publish DocBook 4 to WebHelp With Feedback, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp with Feedback** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

4. Enter the documentation product ID and the documentation version.

When the **DocBook WebHelp with Feedback** transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use_stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `webhelp.copyright` - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
    js = d.createElement(s); js.id = id; js.src =
    "https://connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `l10n.gentext.default.language` - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- `webhelp.search.ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).
- `webhelp.product.id` - This parameter specifies a short name for the documentation target, or product (for example, `mobile-phone-user-guide`, `hvac-installation-guide`). You can deploy documentation for multiple products on the same server.



Note: The following characters are not allowed in the value of this parameter: `< > / \ ' () { }`
`= ; * % + &`

- `webhelp.product.version` - This parameter specifies the documentation version. New comments are bound to this version. Multiple documentation versions can be deployed on the same server.



Note: The following characters are not allowed in the value of this parameter: `< > / \ ' () { }`
`= ; * % + &/>`

For further information about all the DocBook transformation parameters, go to
<http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

To watch our video demonstration about the feedback-enabled WebHelp system, go to
http://oxygentools.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Mobile Output

To generate a mobile WebHelp system from your DocBook 4 document, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp - Mobile** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use.stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `webhelp.copyright` - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- `l10n.gentext.default.language` - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
    js = d.createElement(s); js.id = id; js.src =
    "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.

When the **DocBook WebHelp - Mobile** transformation is complete, the output is automatically opened in your default browser.

DocBook to PDF Output Customization

Main steps for customization of PDF output generated from DocBook XML documents.

When the default layout and output look of the DocBook to PDF transformation need to be customized, the following main steps should be followed. In this example a company logo image is added to the front matter of a book. Other types of customizations should follow some similar steps.

1. Create a custom version of the DocBook title spec file.

You should start from a copy of the file

[`OXYGEN_DIR`] / `frameworks/docbook/xsl/fo/titlepage.templates.xml` and customize it. The instructions for the spec file can be found [here](#).

An example of spec file:

```
<t:titlepage-content t:side="recto">
  <mediaobject/>
  <title
    t:named-template="book.verso.title"
    font-size="&hsize2;" 
    font-weight="bold"
    font-family="${title.font.family}"/>
  <corauthor/>
  ...
</t:titlepage-content>
```

2. Generate a new XSLT stylesheet from the title spec file from the previous step.

Apply [OXYGEN_DIR]/frameworks/docbook/xsl/template/titlepage.xsl to the title spec file.
The result is an XSLT stylesheet, let's call it mytitlepages.xsl.

3. Import mytitlepages.xsl in a *DocBook customization layer*.

The customization layer is the stylesheet that will be applied to the XML document. The mytitlepages.xsl should be imported with an element like:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert logo image in the XML document.

The path to the logo image must be inserted in the book/info/mediaobject element of the XML document.

5. Apply the customization layer to the XML document.

A quick way is duplicating the transformation scenario **DocBook PDF** that comes with Oxygen and set the customization layer in *the XSL URL property of the scenario*.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their .otf file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:

```
@font-face {
    font-family: "MyFont";
    font-weight: bold;
    font-style: normal;
    src: url(fonts/MyFont.otf);
}
```

- tell the CSS where this font is used. To set it as default for h1 elements, use the font-family rule as in the following example:

```
h1 {
    font-size: 20pt;
    margin-bottom: 20px;
    font-weight: bold;
    font-family: "MyFont";
    text-align: center;
}
```

- in your DocBook to EPUB transformation, set the epub.embedded.fonts parameter to fonts/MyFont.otf. If you need to provide more files, use comma to separate their file paths.



Note: The html.stylesheet parameter allows you to include a custom CSS in the output EPUB.

DocBook 4 Templates

Default templates are available in the *New File* wizard. You can use them to create a skeletal form of a DocBook 4 book or article. These templates are stored in the [OXYGEN_DIR]/frameworks/docbook/templates/DocBook 4 folder.

Here are some of the DocBook 4 templates available when creating *new documents from templates*.

- Article
- Article with MathML
- Article with SVG
- Article with XInclude
- Book
- Book with XInclude
- Chapter
- Section
- Set of Books

Inserting olink Links in DocBook 5 Documents

An olink is a type of link between two DocBook XML documents.

The olink element is the equivalent for linking outside the current DocBook document. It has the attribute targetdoc for the document ID that contains the target element and the attribute targetptr for the ID (the value of an id or xml:id attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the *Administrator Guide* is a book with the document ID MailAdminGuide and it contains a chapter about user accounts like the following:

```
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...

```

You can form a cross reference to that chapter by adding an olink in the *User Guide* like the following:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

1. Decide what documents are included in the domain for cross referencing.

An ID should be assigned to each document that will be referenced with an olink. Usually it is added as an id or xml:id attribute to the root element of the document. A document ID is a string that is unique for each document in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like ug, ag, and ref or more specific IDs like MailUserGuide, MailAdminGuide, and MailReference.

2. Decide the output hierarchy.

For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element <dir name="directory_name" > in the target database document. In the example from the next step the hierarchy is documentation/guides/mailuser, documentation/guides/mailadmin, documentation/guides/reference.

3. Create the target database document.

Each collection of documents has a master target database document that is used to resolve all olinks from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically. An example is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE targetset [
<!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
<!ENTITY agtargets SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>
  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
                    baseuri="userguide.html">
            &ugtargets;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargets;
          </document>
        </dir>
      </dir>
    </dir>
  </sitemap>
</targetset>
```

```

</dir>
<dir name="reference">
  <dir name="mailref">
    <document targetdoc="MailReference">
      &reftargets;
    </document>
  </dir>
</dir>
</sitemap>
</targetset>

```

An example of a target.db file:

```

<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">

  <ttx>Administering User Accounts</ttx>
  <xreftext>How to administer user accounts</xreftext>
  <div element="part" href="#d5e4" number="I">
    <ttx>First Part</ttx>
    <xreftext>Part I, "First Part"</xreftext>
    <div element="chapter" href="#d5e6" number="1">
      <ttx>Chapter Title</ttx>
      <xreftext>Chapter 1, Chapter Title</xreftext>
      <div element="sect1" href="#src_chapter" number="1"
targetptr="src_chapter">
        <ttx>Section1 Title</ttx>
        <xreftext>xreflabel_here</xreftext>
      </div>
    </div>
  </div>
</div>

```

4. Generate the target data files.

These files are the target.db files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called collect.xref.targets must be set to the value yes. The default name of a target data file is target.db but it can be changed by setting an absolute file path in the XSLT parameter targets.filename.

5. Insert olink elements in the DocBook XML documents.

When a DocBook XML document is edited in Author mode Oxygen XML Editor provides the **Insert OLink** action on the toolbar. This action allows selecting the target of an olink from the list of all possible targets from a specified target database document. In the following image the target database document is called target.xml.

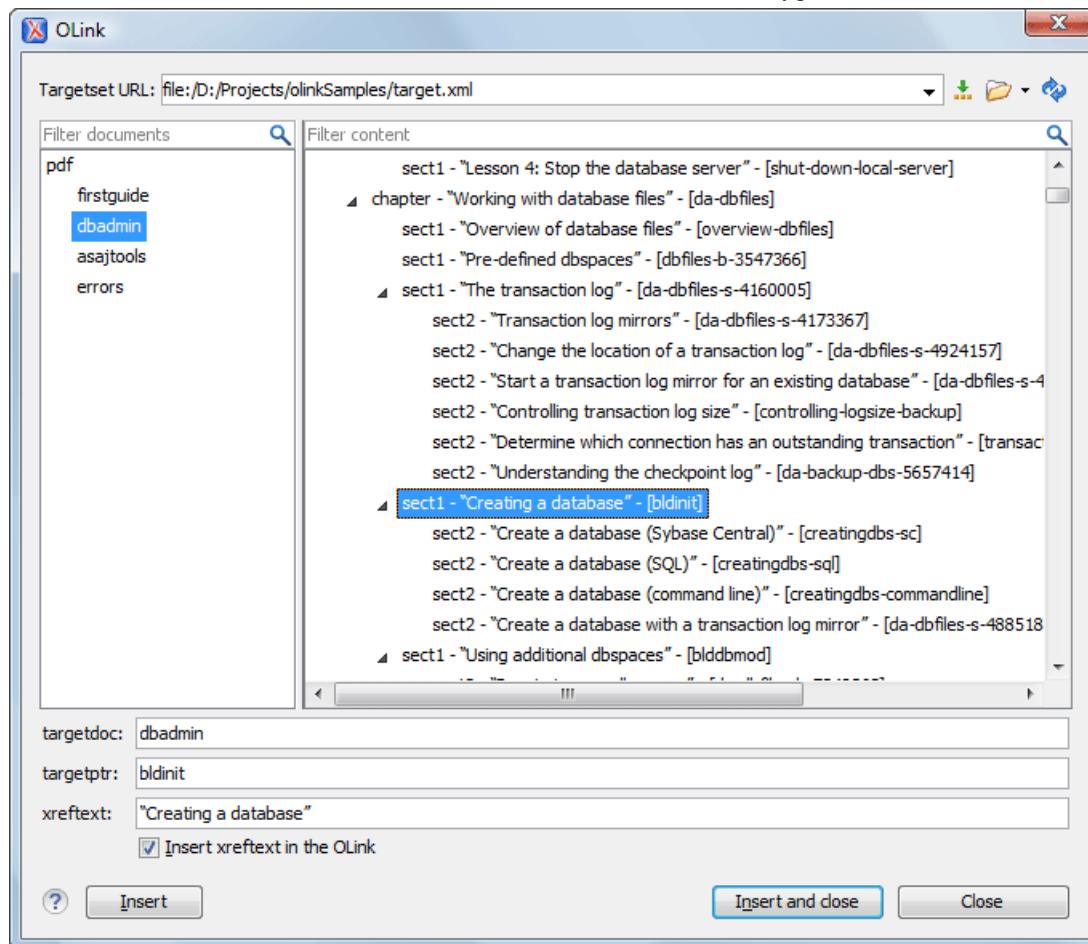


Figure 239: Insert OLink Dialog Box

6. Process each document for output.

That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve `olinks` in the output files using the value of this parameter.

The DocBook 5 Document Type

A file is considered to be a DocBook 5 document when the namespace is `http://docbook.org/ns/docbook`.

The default Relax NG and Schematron schema, `docbookxi.rng`, for these documents is stored in `[OXYGEN_DIR]/frameworks/docbook/5.0/rng/`.

The default CSS files used for rendering DocBook content in **Author** mode is stored in `[OXYGEN_DIR]/frameworks/docbook/css/`.

The default XML catalog, `catalog.xml`, is stored in `[OXYGEN_DIR]/frameworks/docbook/5.0/`.

To watch our video demonstration about editing DocBook documents, go to http://oxygentools.com/demo/DocBook_Editing_in_Author.html.

DocBook 5 Author Actions

The DocBook 5 Author actions are the same as the [DocBook 4 actions](#), with the following exception:

Dragging a file from [the Project view](#) or [DITA Maps Manager view](#) and dropping it into a DocBook 5 document that is edited in **Author** mode, creates a link to the dragged file (the `link` DocBook element) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for

example) and dropping it into a DocBook 5 document inserts an image element (the `inlinemediaobject` DocBook element with an `imagedata` child element) at the drop location, similar to the  **Insert Image Reference** toolbar action.

DocBook 5 Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, EPUB, and EPUB 3.

WebHelp Output

DocBook 5 documents can be transformed into WebHelp systems, such as:

WebHelp Output

To publish DocBook 5 to WebHelp, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp** transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use_stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `webhelp.copyright` - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
      js = d.createElement(s); js.id = id; js.src =
      "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
      fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `l10n_gentext.default.language` - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- `webhelp.search.ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).

WebHelp With Feedback Output

To publish DocBook 5 to WebHelp With Feedback, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp with Feedback** scenario from the **DocBook 5** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

When the **DocBook WebHelp with Feedback** transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use_stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `webhelp_copyright` - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- `webhelp_footer_file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
      js = d.createElement(s); js.id = id; js.src =
      "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
      fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp_footer_include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp_footer_file` parameter has a value, then the content of that file is used as footer. If the `webhelp_footer_file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `l10n_gentext_default_language` - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- `webhelp_logo_image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp_logo_image_target_url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- `webhelp_search_ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).
- `webhelp_product_id` - This parameter specifies a short name for the documentation target, or product (for example, `mobile-phone-user-guide`, `hvac-installation-guide`). You can deploy documentation for multiple products on the same server.



Note: The following characters are not allowed in the value of this parameter: `< > / \ ' () { }`
`= ; * % + &`

- `webhelp_product_version` - This parameter specifies the documentation version. New comments are bound to this version. Multiple documentation versions can be deployed on the same server.



Note: The following characters are not allowed in the value of this parameter: `< > / \ ' () { }`
`= ; * % + &/>`

For further information about all the DocBook transformation parameters, go to
<http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

To watch our video demonstration about the feedback-enabled WebHelp system, go to

http://oxygengxml.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Mobile Output

To generate a mobile WebHelp system from your DocBook 5 document, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp - Mobile** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

To further customize the out-of-the-box transformation, you can edit its parameters:

- **use.stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- **webhelp.copyright** - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- **110n.gentext.default.language** - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return;
      js = d.createElement(s); js.id = id; js.src =
      //connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
      fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk'); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- **webhelp.footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the **webhelp.footer.file** parameter has a value, then the content of that file is used as footer. If the **webhelp.footer.file** has no value then the default Oxygen footer is inserted in each WebHelp page.

When the **DocBook WebHelp - Mobile** transformation is complete, the output is automatically opened in your default browser.

DocBook to PDF Output Customization

Main steps for customization of PDF output generated from DocBook XML documents.

When the default layout and output look of the DocBook to PDF transformation need to be customized, the following main steps should be followed. In this example a company logo image is added to the front matter of a book. Other types of customizations should follow some similar steps.

1. Create a custom version of the DocBook title spec file.

You should start from a copy of the file

`[OXYGEN_DIR]/frameworks/docbook/xsl/fo/titlepage.templates.xml` and customize it. The instructions for the spec file can be found [here](#).

An example of spec file:

```
<t:titlepage-content t:side="recto">
  <mediaobject/>
  <title
    t:named-template="book.verso.title"
```

```

    font-size="&hsize2;
    font-weight="bold";
    font-family="${title.font.family}"/>
<corpauthor/>
...
</t:titlepage-content>
```

- Generate a new XSLT stylesheet from the title spec file from the previous step.

Apply [OXYGEN_DIR]/frameworks/docbook/xsl/template/titlepage.xsl to the title spec file. The result is an XSLT stylesheet, let's call it mytitlepages.xsl.

- Import mytitlepages.xsl in a *DocBook customization layer*.

The customization layer is the stylesheet that will be applied to the XML document. The mytitlepages.xsl should be imported with an element like:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

- Insert logo image in the XML document.

The path to the logo image must be inserted in the book/info/mediaobject element of the XML document.

- Apply the customization layer to the XML document.

A quick way is duplicating the transformation scenario **DocBook PDF** that comes with Oxygen and set the customization layer in *the XSL URL property of the scenario*.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their .otf file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:

```
@font-face {
  font-family: "MyFont";
  font-weight: bold;
  font-style: normal;
  src: url(fonts/MyFont.otf);
}
```

- tell the CSS where this font is used. To set it as default for h1 elements, use the font-family rule as in the following example:

```
h1 {
  font-size: 20pt;
  margin-bottom: 20px;
  font-weight: bold;
  font-family: "MyFont";
  text-align: center;
}
```

- in your DocBook to EPUB transformation, set the epub.embedded.fonts parameter to fonts/MyFont.otf. If you need to provide more files, use comma to separate their file paths.



Note: The html.stylesheet parameter allows you to include a custom CSS in the output EPUB.

DocBook 5 Templates

Default templates are available in the *New File wizard* and can be used for easily creating a skeletal form of a DocBook 5 book or article. These templates are stored in the

[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 5 folder.

Here are some of the DocBook 5 templates available when creating *new documents from templates*.

- Article**;
- Article with MathML**;
- Article with SVG**;
- Article with XInclude**;

- **Book;**
- **Book with XInclude;**
- **Chapter;**
- **Section;**
- **Set of Books.**

Inserting olink Links in DocBook 5 Documents

An `olink` is a type of link between two DocBook XML documents.

The `olink` element is the equivalent for linking outside the current DocBook document. It has the attribute `targetdoc` for the document ID that contains the target element and the attribute `targetptr` for the ID (the value of an `id` or `xml:id` attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the *Administrator Guide* is a book with the document ID `MailAdminGuide` and it contains a chapter about user accounts like the following:

```
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...
```

You can form a cross reference to that chapter by adding an `olink` in the *User Guide* like the following:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

1. Decide what documents are included in the domain for cross referencing.

An ID should be assigned to each document that will be referenced with an `olink`. Usually it is added as an `id` or `xml:id` attribute to the root element of the document. A document ID is a string that is unique for each document in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like `ug`, `ag`, and `ref` or more specific IDs like `MailUserGuide`, `MailAdminGuide`, and `MailReference`.

2. Decide the output hierarchy.

For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element `<dir name="directory_name">` in the target database document. In the example from the next step the hierarchy is `documentation/guides/mailuser`, `documentation/guides/mailadmin`, `documentation/guides/reference`.

3. Create the target database document.

Each collection of documents has a master target database document that is used to resolve all `olinks` from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically. An example is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE targetset [
  <!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
  <!ENTITY agtargets SYSTEM "file:///doc/adminguide/target.db">
  <!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>
  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
```

```

<document targetdoc="MailUserGuide"
           baseuri="userguide.html">
  &ugtargets;
</document>
</dir>
<dir name="mailadmin">
  <document targetdoc="MailAdminGuide">
    &agttargets;
  </document>
</dir>
</dir>
<dir name="reference">
  <dir name="mailref">
    <document targetdoc="MailReference">
      &reftargets;
    </document>
  </dir>
</dir>
</dir>
</sitemap>
</targetset>

```

An example of a target.db file:

```

<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">

  <ttx>Administering User Accounts</ttx>
  <xreftext>How to administer user accounts</xreftext>
  <div element="part" href="#d5e4" number="I">
    <ttx>First Part</ttx>
    <xreftext>Part I, "First Part"</xreftext>
    <div element="chapter" href="#d5e6" number="1">
      <ttx>Chapter Title</ttx>
      <xreftext>Chapter 1, Chapter Title</xreftext>
      <div element="sect1" href="#src_chapter" number="1"
targetptr="src_chapter">
        <ttx>Section1 Title</ttx>
        <xreftext>xreflabel_here</xreftext>
      </div>
    </div>
  </div>
</div>

```

4. Generate the target data files.

These files are the target.db files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called collect.xref.targets must be set to the value yes. The default name of a target data file is target.db but it can be changed by setting an absolute file path in the XSLT parameter targets.filename.

5. Insert olink elements in the DocBook XML documents.

When a DocBook XML document is edited in Author mode provides the **Insert OLink** action on the toolbar. This action allows selecting the target of an olink from the list of all possible targets from a specified target database document. In the following image the target database document is called target.xml.

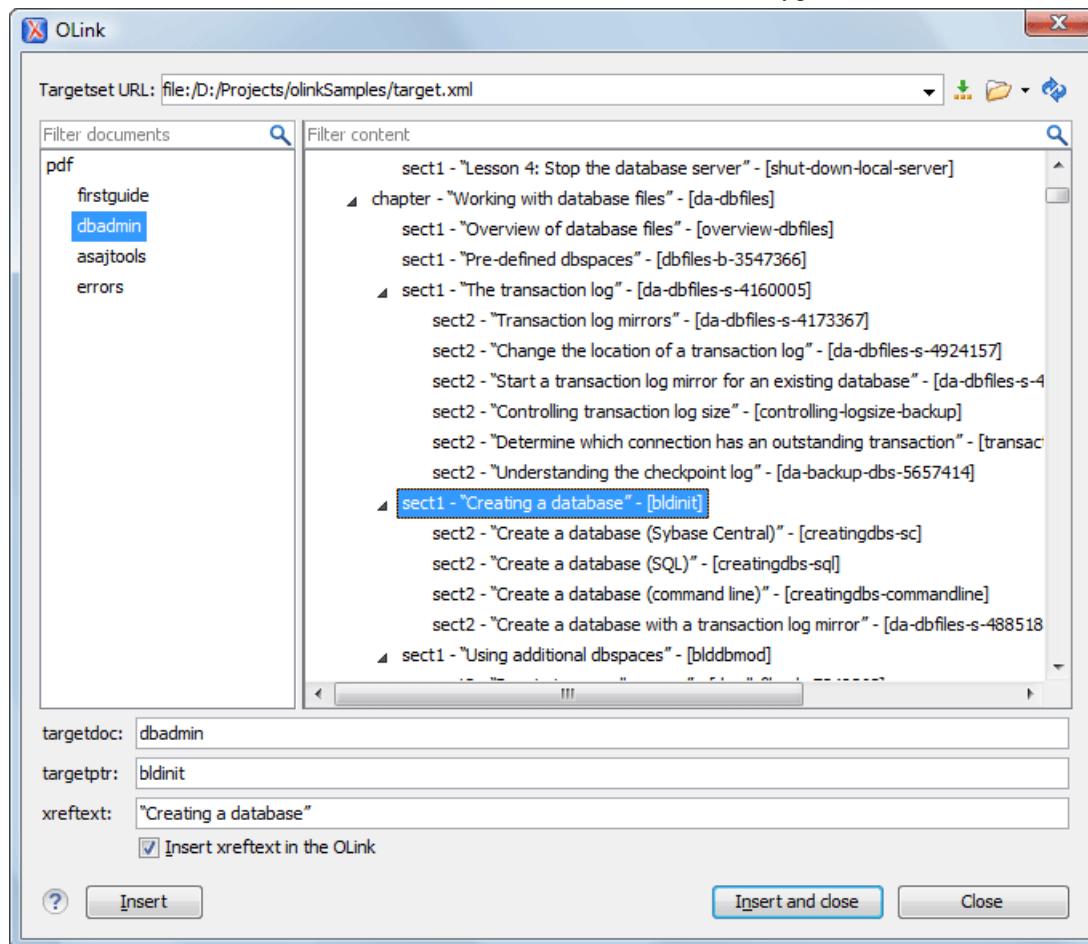


Figure 240: Insert OLink Dialog Box

6. Process each document for output.

That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve `olinks` in the output files using the value of this parameter.

The DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture oriented to authoring, producing, and delivering technical information. It divides content into small, self-contained topics that you can reuse in various deliverables. The extensibility of DITA permits organizations to define specific information structures while still using standard tools to work with them. Oxygen XML Editor provides schema-driven (DTD, RNG, XSD) templates for DITA documents.

A file is considered to be a DITA topic document when one of the following conditions are true:

- The root element name is one of the following: `concept`, `task`, `reference`, `dita`, or `topic`.
- The PUBLIC ID of the document is a PUBLIC ID for the elements listed above.
- The root element of the file has an attribute named `DITAArchVersion` for the `"http://dita.oasis-open.org/architecture/2005/"` namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option is enabled from the [Document Type Association preferences page](#).

The default schemas used for DITA topic documents are stored in

`[OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/` or
`[OXYGEN_DIR]/frameworks/dita/DITA-OT/schema/`.

The default CSS files used for rendering DITA content in **Author** mode are stored in [OXYGEN_DIR]/frameworks/dita/css/.

The default catalogs for the DITA topic document type are as follows:

- [OXYGEN_DIR]/frameworks/dita/catalog.xml
- [OXYGEN_DIR]/frameworks/dita/DITA-OT/catalog-dita.xml
- [OXYGEN_DIR]/frameworks/dita/plugin/catalog.xml
- [OXYGEN_DIR]/frameworks/dita/styleguide/catalog.xml

DITA Author Actions

The following default actions are available in the **DITA (Author Custom Actions)** toolbar:

B Bold

Surrounds the selected text with a **b** tag. You can use this action on multiple non-contiguous selections.

I Italic

Surrounds the selected text with an **i** tag. You can use this action on multiple non-contiguous selections.

U Underline

Surrounds the selected text with a **u** tag. You can use this action on multiple non-contiguous selections.

Link Actions Drop-Down List

The following link actions are available from this list:

- **Cross Reference** - Depending on the context where it is invoked, the action inserts one of the following two elements:
 - **xref** element, with the **format** attribute set to **dita**
 - **fragref** element, which is a specialization of the **xref** element



Note: Both elements point to their target using the **href** attribute

The referenced target is selected in a dialog box that lists all the IDs extracted from the selected file. When you select an ID, you can preview the content in the **Preview** tab or the XML source in the **Source** tab. In case you have a large number of IDs in the target document, use the **Filter** field to search through the IDs.

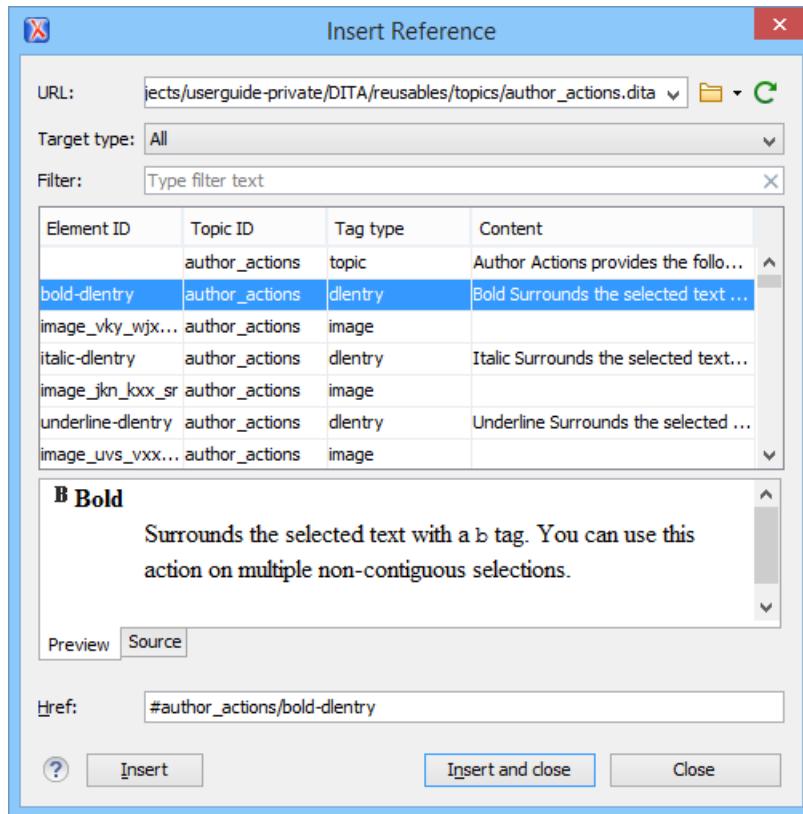


Figure 241: Insert a Cross Reference in a DITA Document



Note: The **Insert Reference** dialog box is not modal. The dialog box is closed automatically in case you switch to a different editor.

- **Key Reference** - Inserts a user specified element with the value of the keyref attribute set to a specific key name. As stated in the DITA 1.2 specification, keys are defined at map level and referenced afterwards. You are able to select the target of the keyref element in the **Insert Key Reference** dialog box.



Note: The **Insert Key Reference** dialog box presents the list of keys available in the current DITA Map. If the DITA Map is not opened in the **DITA Maps Manager** view, the **Insert Key Reference** dialog box does not display any keys.

You can also reference elements at sub-topic level by pressing the **Sub-topic** button and choosing the target.

All keys which are presented in the dialog box are gathered from the current opened DITA map. Elements which have the keyref attribute set are displayed as links. The current opened DITA map is also used to resolve references when navigating keyref links in the Author mode. Image elements which use key references are rendered as images.

- **File Reference** - Inserts an xref element with the value of attribute format set to xml.
- **Web Link** - Inserts an xref element with the value of attribute format set to html, and scope set to external.
- **Related Link to Topic** - Inserts a link element inside a related-links parent.
- **Related Link to File** - Inserts a link element with the format attribute set to xml inside a related-links parent.
- **Related Link to Web Page** - Inserts a link element with the attribute format set to html and scope set to external inside a related-links parent.

 **Insert Image Reference**

Opens the **Insert Image** dialog box that allows you to configure the properties of an image to be inserted into a DITA document at the caret position.

§  **Insert Section Drop-Down List**

The following link actions are available from this list:

-  **Insert Section** - Inserts a new section / step in the document, depending on the current context. A new section will be inserted in either one of the following contexts:
 - section context, when the value of `class` attribute of the current element or one of its ancestors contains `topic` or `section`.
 - topic's body context, when the value of `class` attribute of the current element contains `topic/body`.
- A new step will be inserted in either one of the following contexts:
 - task step context, when the value of `class` attribute of the current element or one of its ancestors contains `task/step`.
 - task steps context, when the value of `class` attribute of the current element contains `task/steps`.
-  **Insert Concept** - Inserts a new concept. Concepts provide background information that users must know before they can successfully work with a product or interface. This action is available in one of the following contexts:
 - concept context, one of the current element ancestors is a `concept`. In this case an empty `concept` will be inserted after the current `concept`.
 - concept or DITA context, current element is a `concept` or `dita`. In this case an empty `concept` will be inserted at current caret position.
 - DITA topic context, current element is a `topic` child of a `dita` element. In this case an empty `concept` will be inserted at current caret position.
 - DITA topic context, one of the current element ancestors is a DITA `topic`. In this case an empty `concept` will be inserted after the first `topic` ancestor.
-  **Insert Task** - Inserts a new task. Tasks are the main building blocks for task-oriented user assistance. They generally provide step-by-step instructions that will enable a user to perform a task. This action is available in one of the following contexts:
 - task context, one of the current element ancestors is a `task`. In this case an empty `task` will be inserted after the last child of the first `concept`'s ancestor.
 - task context, the current element is a `task`. In this case an empty `task` will be inserted at current caret position.
 - topic context, the current element is a `dita topic`. An empty `task` will be inserted at current caret position.
 - topic context, one of the current element ancestors is a `dita topic`. An empty `task` will be inserted after the last child of the first ancestor that is a `topic`.
-  **Insert Topic** -
-  **Insert Reference** - Inserts a new reference in the document. A reference is a top-level container for a reference topic. This action is available in one of the following contexts:
 - reference context - one of the current element ancestors is a `reference`. In this case an empty `reference` will be inserted after the last child of the first ancestor that is a `reference`.
 - reference or `dita` context - the current element is either a `dita` or a `reference`. An empty `reference` will be inserted at caret position.
 - topic context - the current element is `topic` descendant of `dita` element. An empty `reference` will be inserted at caret position.
 - topic context - the current element is descendant of `dita` element and descendant of `topic` element. An empty `reference` will be inserted after the last child of the first ancestor that is a `topic`.

Insert a new paragraph

Insert a new paragraph at current cursor position.

Insert DITA Content Reference

Inserts a content reference at the caret position.

The DITA `conref` attribute provides a mechanism for reuse of content fragments. The `conref` attribute stores a reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element. See [here](#) for more details.

Oxygen XML Editor *displays the referenced content* of a DITA `conref` if it can resolve it to a valid resource. If you have URI's instead of local paths in the XML documents and your DITA OT transformation needs an XML catalog to map the URI's to local paths you have to [add the catalog to Oxygen XML Editor](#). If the URI's can be resolved the referenced content will be displayed in Author mode and in the transformation output.

A content reference is inserted with the action **Insert a DITA Content Reference** available on the toolbar **Author custom actions** and on the menu **DITA > Insert**.

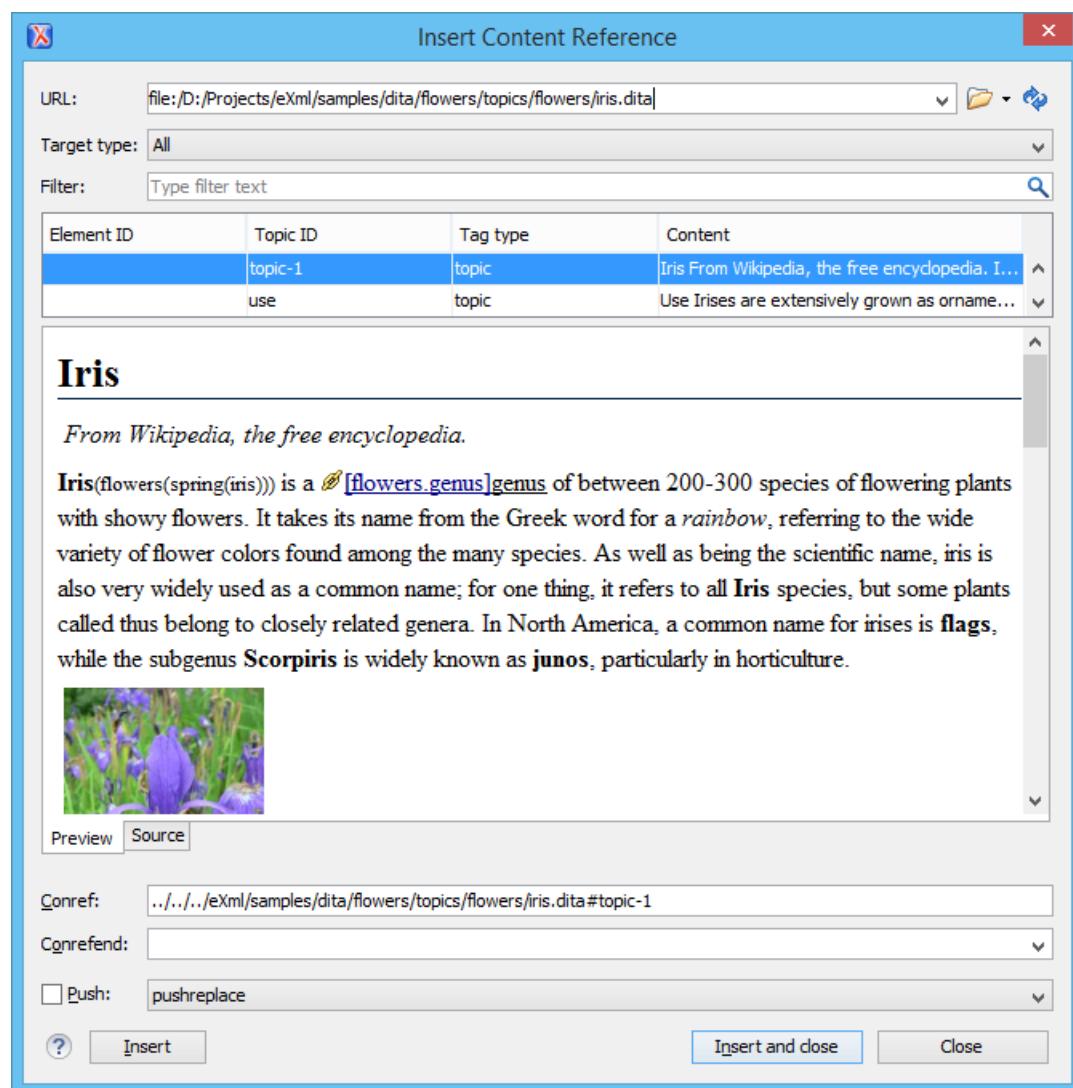


Figure 242: Insert Content Reference Dialog Box



Note: The **Insert Content Reference** dialog box is not modal. The dialog box is closed automatically in case you switch to a different editor.

In the URL chooser you set the URL of the file from which you want to reuse content. Depending on the **Target type** filter you will see a tree of elements which can be referenced (which have ID's). For each element the XML content is shown in the preview area. The **Conref** value is computed automatically for the selected tree element. After pressing **Insert**, an element with the same name as the target element and having the attribute **conref** with the value specified in the **Conref** value field will be inserted at caret position.

According to the DITA 1.2 specification the **conrefend** attribute can be used to specify content reference ranges. This is a very useful feature when referencing multiple consecutive steps or list items. If you use multiple contiguous sibling selection the **conrefend** value will also be set to the value of the last selected ID path. Oxygen XML Editor will present the entire referenced range as read-only content.



Insert Content Key Reference

Inserts a content key reference at the caret position.

As stated in the DITA 1.2 specification the **conkeyref** attribute provides a mechanism for reuse of content fragments similar with the **conref** mechanism. Keys are defined at map level which can be referenced using **conkeyref**. The **conkeyref** attribute contains a key reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content key reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element.

Oxygen XML Editor *displays the key referenced content* of a DITA **conkeyref** if it can resolve it to a valid resource in the context of the current opened DITA map.

A content key reference is inserted with the action **Insert a DITA Content Key Reference** available on the toolbar **Author custom actions** and on the menu **DITA > Insert**.

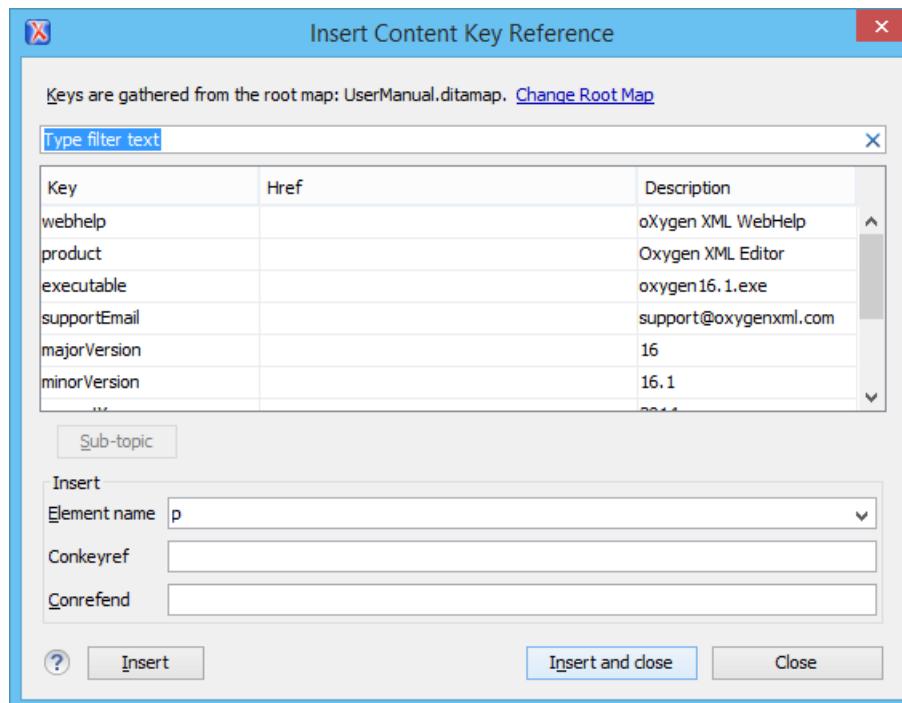


Figure 243: Insert Content Key Reference Dialog Box



Note: The **Insert Content Key Reference** dialog box is not modal. The dialog box is closed automatically in case you switch to a different editor.

To reference target elements at sub-topic level just press the **Sub-topic** button and choose the target.

According to the DITA 1.2 specification the `conrefend` attribute can be used to specify content reference ranges. This is a very useful feature when referencing multiple consecutive steps or list items. If you use multiple contiguous sibling selection for IDs at sub-topic level the `conrefend` value will also be set to the value of the last selected ID path. Oxygen XML Editor will present the entire referenced range as read-only content.

 **Important:** All keys which are presented in the dialog box are gathered from the current opened DITA map. Elements which have the `conkeyref` attribute set are displayed by default with the target content expanded. The current opened DITA map is also used to resolve references when navigating `conkeyref` links in the Author mode.

Insert a step or list item

Inserts a new list or step item in the current list type.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also automatically inserted by default.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also automatically inserted by default.

Sort

Sorts a table selection.

Insert Table

Opens a dialog box that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert Row

Inserts a new table row with empty cells. This action is available when the caret is positioned inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, Oxygen XML Editor a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at caret position.

Delete Row

Deletes the table row located at caret position.

Edit Table Properties

Opens the **Table properties** dialog box that allows you to configure properties of a table (such as frame borders).

Table Join/Split Drop-Down List

The following link actions are available from this list:

-  **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
-  **Join Cell Above** - Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.
-  **Join Cell Below** - Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.



Note: When you use  **Join Cell Above** and  **Join Cell Below**, Oxygen XML Editor deletes the source row if it remains empty. The cells that span over multiple rows are also updated.

-  **Split Cell To The Left** - Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column. Oxygen XML Editor decreases the column span of the source cell with one.
-  **Split Cell To The Right** - Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column. Oxygen XML Editor decreases the column span of the source cell with one.
-  **Split Cell Above** - Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row. Oxygen XML Editor decreases the column span of the source cell with one.
-  **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row. Oxygen XML Editor decreases the column span of the source cell with one.

In addition, the following default actions are available from the DITA menu:

Refresh references

You can use this action to manually trigger a refresh and update of all referenced resources.

In addition, the following default actions are available from the contextual menu:

Style Guide

Opens the **DITA Style Guide Best Practices for Authors** in your browser and displays a topic that is relevant to the element at the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu), in the **DITA** menu, and in some of the documentation tips that are displayed by the **Content Completion Assistant**.

Browse reference manual

Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu) and in the documentation tip displayed by the **Content Completion Assistant**.

Paste special > Paste as content reference

Available on the contextual menu of Author editor for any topic file, this operation inserts a content reference (a DITA element with a `conref` attribute) to the DITA XML element from the clipboard. An entire DITA XML element with an ID attribute must be present in the clipboard when the action is invoked. The `conref` attribute will point to this ID value.

Paste special > Paste as content key reference

Allows you to indirectly reference content using the `conkeyref` attribute. When the DITA content is processed, the key references are resolved using key definitions from DITA maps. To use this action, do the following:

1. Set the `id` attribute of the element holding the content you want to reference.
2. Open the DITA Map in the **DITA Maps Manager** view and make sure that the **Root map** combo box points to the correct map that stores the keys.
3. Right click the topic that holds the content you want to reference, select **Edit Properties**, and enter a value in the **Keys** field.

Paste special > Paste as link

Available on the contextual menu of Author editor for any topic file, this action inserts a `link` element or an `xref` one (depending on the location of the paste operation) that points to the DITA XML element from the clipboard. An entire DITA XML element with an ID attribute must be present in the clipboard when the action is invoked. The `href` attribute of `link/xref` will point to this ID value.

Paste special > Paste as link (keyref)

Inserts a link to the element that you want to reference. To use this action, do the following:

1. Set the `id` attribute of the element that you want to reference.

2. Open the DITA Map in the **DITA Maps Manager** view and make sure that the **Root map** combo box points to the correct map that stores the keys.
3. Right click the topic that holds the content you want to reference, select **Edit Properties**, and enter a value in the **Keys** field.

Replace conref / conkeyref reference with content

Replaces the content reference fragment or the `conkeyref` at caret position with the referenced content. This action is useful when you want to make changes to the content but decide to keep the referenced fragment unchanged.

Insert Equation

Allows you to insert an MathML equation. For more information, see section [Editing MathML Notations](#).

Create Reusable Component

Creates a reusable component from a selected fragment of text. For more information, see [Reusing Content](#).

Insert Reusable Component

Inserts a reusable component at cursor location. For more information, see [Reusing Content](#).

Remove Content Reference

Removes the `conref` attribute of an element. For more information, see [Reusing Content](#).

Add/Edit Content Reference

Add or edit the `conref` attribute of an element. For more information, see [Reusing Content](#).

Generate IDs

This action generates and sets unique IDs for:

- The element at caret position.
- All top-level elements found in the current selection. Additionally, if the selection contains elements from the **DITA > ID Options** list, they will all receive an unique ID



Note: IDs already set are preserved.

The action is available both in the contextual menu and in the **DITA** main menu.

ID Options

Action available in the **DITA** main menu, allows you to specify the elements for which Oxygen XML Editor generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported [editor variables](#).

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs when copying content in the same document** option is not checked.

Search References

Finds the references to the `id` attribute value of the selected element in all the topics from the current DITA map (opened in the **DITA Maps Manager** view). The default shortcut of the action is **Ctrl Shift G (Command Shift G on OS X)** and can be changed in the **DITA Topic** document type.

Dragging a file from [the Project view](#) or [DITA Maps Manager view](#) and dropping it into a DITA document that is edited in **Author** mode, creates a link to the dragged file (the `xref` DITA element with the `href` attribute) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into a DITA document inserts an image element (the `image` DITA element with the `href` attribute) at the drop location.

DITA Transformation Scenarios

The following default transformation scenarios are available for DITA Topics:

- **DITA XHTML** - Transforms a DITA topic to XHTML using DITA Open Toolkit.
- **DITA PDF** - Transforms a DITA topic to PDF using the DITA Open Toolkit and the Apache FOP engine.

DITA Templates

The default templates available for DITA topics are stored in `[OXYGEN_DIR]/frameworks/dita/templates/topic` folder. They can be used for easily creating a DITA concept, reference, task or topic.

Here are some of the DITA templates available when creating [new documents from templates](#):

- **Composite** - New DITA Composite
- **Composite with MathML** - New DITA Composite with MathML
- **Concept** - New DITA Concept
- **General Task** - New DITA Task
- **Glossentry** - New DITA Glossentry
- **Glossgroup** - New DITA Glossgroup
- **Machinery Task** - New DITA Machinery Task
- **Reference** - New DITA Reference
- **Task** - New DITA Task
- **Topic** - New DITA Topic
- **Learning Assessment** - New DITA Learning Assessment (learning specialization in DITA 1.2)
- **Learning Content** - New DITA Learning Content (learning specialization in DITA 1.2)
- **Learning Summary** - New DITA Learning Summary (learning specialization in DITA 1.2)
- **Learning Overview** - New DITA Learning Overview (learning specialization in DITA 1.2)
- **Learning Plan** - New DITA Learning Plan (learning specialization in DITA 1.2)
- **Troubleshooting** - Experimental DITA 1.3 troubleshooting specialization

DITA for Publishers topic specialization templates:

- **D4P Article** - New DITA for Publishers article
- **D4P Chapter** - New DITA for Publishers chapter
- **D4P Concept** - New DITA for Publishers concept
- **D4P Conversion Configuration** - New DITA for Publishers conversion configuration
- **D4P Cover** - New DITA for Publishers cover
- **D4P Part** - New DITA for Publishers part
- **D4P Sidebar** - New DITA for Publishers sidebar
- **D4P Subsection** - New DITA for Publishers subsection
- **D4P Topic** - New DITA for Publishers topic

The DITA Map Document Type

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships among the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests for DITA projects.

Maps allow scalable reuse of content across multiple contexts. They can be used by information architects, authors, and publishers to plan, develop, and deliver content.

A file is considered to be a DITA map document when either of the following is true:

- The root element name is one of the following: `map`, `bookmap`.
- The public id of the document is `-//OASIS//DTD DITA Map` or `-//OASIS//DTD DITA BookMap`.
- The root element of the file has an attribute named `class` which contains the value `map/map` and a `DITAArchVersion` attribute from the `http://dita.oasis-open.org/architecture/2005/` namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the [Document Type Detection option page](#) is enabled.

The default schemas used for DITA map documents are stored in `[OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/` or `[OXYGEN_DIR]/frameworks/dita/DITA-OT/schema/`.

The default CSS files used for rendering DITA content in **Author** mode are stored in [OXYGEN_DIR]/frameworks/dita/css/.

The default catalogs for the DITA map document type are as follows:

- [OXYGEN_DIR]/frameworks/dita/catalog.xml
- [OXYGEN_DIR]/frameworks/dita/DITA-OT/catalog-dita.xml

DITA Map Author Actions

When a DITA map is opened in the editor, the following default actions are available in the **DITA** submenu of the main menu, and in the **Author custom actions** toolbar:

Insert New Topic

Creates a new topic and inserts a reference to it at the caret position.

Insert Topic Reference

Inserts a reference to a topic.

Insert Content Reference

Inserts a content reference at the caret position.

Insert Content Key Reference

Inserts a content key reference at the caret position.

Insert Topic Heading

Inserts a topic heading at the caret position.

Insert Topic Group

Inserts a topic group at the caret position.

Insert Relationship Table

Opens a dialog box that allows you to configure the relationship table to be inserted. The dialog box allows the user to configure the number of rows and columns of the relationship table, if the header will be generated and if the title will be added.

Relationship Table Properties

Allows you to change the properties of rows in relationship tables.

Insert Row

Inserts a new table row with empty cells. The action is available when the caret position is inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.

Delete Column

Deletes the table column where the caret is located.

Delete Row

Deletes the table row where the caret is located.

Dragging a file from *the Project view* or *DITA Maps Manager view* and dropping it into a DITA map document that is edited in **Author** mode creates a link to the dragged file (a `topicref` element, `chapter`, `part`, etc.) at the drop location.

DITA Map Transformation Scenarios

The following default transformations are available:

- Predefined transformation scenarios allow you to transform a DITA Map to PDF, ODF, XHTML, WebHelp, EPUB, and CHM files.

- **Run DITA-OT Integrator** - Use this transformation scenario if you want to integrate a DITA-OT plugin. This scenario runs an ANT task that integrates all the plug-ins from the DITA-OT/plugins directory.
- **DITA Map Metrics Report** - Use this transformation scenario if you want to generate a DITA Map statistics report containing information such as:
 - the number of processed maps and topics
 - content reuse percentage
 - number of elements, attributes, words, and characters used in the entire DITA Map structure
 - DITA conditional processing attributes used in the DITA Maps
 - words count
 - information types such as number of containing maps, bookmaps, or topics

Many more output formats are available by clicking the **New** button. The transformation process relies on the DITA Open Toolkit.

WebHelp Output

DITA Maps can be transformed into WebHelp systems, such as:

WebHelp Output

To publish a DITA Map to WebHelp:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DITA Map WebHelp** scenario from the **DITA Map** section.
3. Click **Apply associated**.

When the **DITA Map WebHelp** transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use_stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `clean_output` - Deletes all files from the output folder before the transformation is performed (only `no` and `yes` values are valid and the default value is `no`).
- `webhelp.copyright` - Adds a small copyright text that appears at the end of the Table of Contents.
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
    js = d.createElement(s); js.id = id; js.src =
    "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: '`yes`', '`no`'. If set to '`no`' no footer is added to the WebHelp pages. If set to '`yes`' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

- `webhelp.search.ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).
- `args.default.language` - If the language is not detected in the DITA map, this parameter is used. The default sample value is `en-us`.
- `webhelp.search.japanese.dictionary` - The file path of the user dictionary that will be used by the Kuromoji morphological indexer that is used for indexing Japanese content in the WebHelp pages.

WebHelp With Feedback Output

To publish a DITA Map as WebHelp with Feedback:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DITA Map WebHelp with Feedback** scenario from the **DITA Map** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

When the **DITA Map WebHelp with Feedback** transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use.stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `clean.output` - Deletes all files from the output folder before the transformation is performed (only `no` and `yes` values are valid and the default value is `no`).
- `webhelp.copyright` - Adds a small copyright text that appears at the end of the Table of Contents.
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
      return;
    js = d.createElement(s); js.id = id; js.src =
    "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: '`yes`', '`no`'. If set to '`no`' no footer is added to the WebHelp pages. If set to '`yes`' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- `webhelp.search.ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).
- `args.default.language` - If the language is not detected in the DITA map, this parameter is used. The default sample value is `en-us`.
- `webhelp.search.japanese.dictionary` - The file path of the user dictionary that will be used by the Kuromoji morphological indexer that is used for indexing Japanese content in the WebHelp pages.

To watch our video demonstration about the feedback-enabled WebHelp system, go to

http://oxygentools.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Mobile Output

To generate a mobile WebHelp system from your DITA Map:

1. From the **DITA Maps Manager** view click  **Configure Transformation Scenarios**.
2. Select the **DITA Map WebHelp - Mobile** transformation scenario.
3. Click **Apply associated**.

When the **DITA Map WebHelp - Mobile** transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- **use_stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- **webhelp_copyright** - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- **webhelp_indexer.language** - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.
- **webhelp_footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
        <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
            return;
        js = d.createElement(s); js.id = id; js.src =
        "https://connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
        fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk');
    </script>
    <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- **webhelp_footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the **webhelp_footer.file** parameter has a value, then the content of that file is used as footer. If the **webhelp_footer.file** has no value then the default Oxygen footer is inserted in each WebHelp page.
- **args.default.language** - If the language is not detected in the DITA map, this parameter is used. The default sample value is `en-us`.
- **webhelp_search.japanese.dictionary** - The file path of the user dictionary that will be used by the Kuromoji morphological indexer that is used for indexing Japanese content in the WebHelp pages.

Once Oxygen XML Editor finishes the transformation process, the output is automatically opened in your default browser.

How to Localize the Interface of WebHelp Output

Static labels that are used in the WebHelp output are kept in translation files in the `[OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization` folder. By default, the DITA-OT folder is `[OXYGEN_DIR]/frameworks/dita/DITA-OT`, or possibly elsewhere if you are using a different DITA-OT distribution. Translation files have the `strings-lang1-lang2.xml` name format, where `lang1` and `lang2` are ISO language codes. For example, the US English text is kept in the `strings-en-us.xml` file.

Follow these steps to localize the interface of the WebHelp output:

1. Look for the `strings-[lang1]-[lang2].xml` file in `[OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization` (for example, the Canadian French file would be: `strings-fr-ca.xml`). If it does not exist, create one starting from `strings-en-us.xml`.

2. Translate all the labels from the above language file. Labels are stored in XML elements that have the following format: <str name="Label name">Caption</str>.
3. Make sure that the new XML file that you created in the previous two steps is listed in the file `[OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization/strings.xml`. In our example for the Canadian French file, it should be listed as: <lang xml:lang="fr-ca" filename="strings-fr-ca.xml" />
4. Edit the **DITA Map WebHelp/DITA Map WebHelp with Feedback** transformation scenario and set the args.default.language parameter to the code of the language you want to localize (for example, *fr-ca* for Canadian French).
5. Run the transformation scenario to produce the WebHelp output.

Support for Right-to-Left (RTL) Oriented Languages

To activate support for RTL languages, edit the DITA Map and set the `xml:lang` attribute on its root element (`map`). The corresponding attribute value can be set for following RTL languages:

- ar-eg - Arabic
- he-il - Hebrew
- ur-pk - Urdu

WebHelp Search Engine Optimization

A DITA WebHelp transformation scenario can be configured to produce a `sitemap.xml` file that is used by search engines to aid crawling and indexing mechanisms. A `sitemap` lists all pages of a WebHelp system and allows webmasters to provide additional information about each page, such as the date it was last updated, change frequency, and importance of each page in relation to other pages in your WebHelp deployment.

The structure of the `sitemap.xml` file looks like this:

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/topics/introduction.html</loc>
    <lastmod>2014-10-24</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
  <url>
    <loc>http://www.example.com/topics/care.html#care</loc>
    <lastmod>2014-10-24</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
  .
  .
</urlset>
```

Each page has a `<url>` element structure containing additional information, such as:

- `loc` - the URL of the page. This URL must begin with the protocol (such as `http`), if required by your web server. It is constructed from the value of the `webhelp.sitemap.base.url` parameter from the transformation scenario and the relative path to the page (collected from the `href` attribute of a `topicref` element in the DITA map).



Note: The value must have less than 2,048 characters.

- `lastmod` - the date when the page was last modified. The date format is `YYYY-MM-DD`.
- `changefreq` - indicates how frequently the page is likely to change. This value provides general information to assist search engines, but may not correlate exactly to how often they crawl the page. Valid values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`. The first time the `sitemap.xml` file is generated, the value is set based upon the value of the `webhelp.sitemap.change.frequency` parameter in the DITA WebHelp transformation scenario. You can change the value in each `url` element by editing the `sitemap.xml` file.



Note: The value `always` should be used to describe documents that change each time they are accessed. The value `never` should be used to describe archived URLs.

- **priority** - the priority of this page relative to other pages on your site. Valid values range from 0.0 to 1.0. This value does not affect how your pages are compared to pages on other sites. It only lets the search engines know which pages you deem most important for the crawlers. The first time the `sitemap.xml` file is generated, the value is set based upon the value of the `webhelp.sitemap.priority` parameter in the DITA WebHelp transformation scenario. You can change the value in each `url` element by editing the `sitemap.xml` file.

 **Note:** `lastmod`, `changefreq`, and `priority` are optional elements.

Creating and Editing the `sitemap.xml` File

Follow these steps to produce a `sitemap.xml` file for your WebHelp system, which can then be edited to fine-tune search engine optimization:

1. **Edit** the transformation scenario you currently use for obtaining your WebHelp output. This opens the **Edit DITA Scenario** dialog.
2. Open the **Parameters** tab and set a value for the following parameters:
 - `webhelp.sitemap.base.url` - the URL of the location where your WebHelp system is deployed

 **Note:** This parameter is required in order for Oxygen XML Editor to generate the `sitemap.xml` file.

 - `webhelp.sitemap.change.frequency` - how frequently the WebHelp pages are likely to change (accepted values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`)
 - `webhelp.sitemap.priority` - the priority of each page (value ranging from 0.0 to 1.0)
3. Execute the transformation scenario.
4. Look for the `sitemap.xml` file in the transformation's output folder. Edit the file to fine-tune the parameters of each page, according to your needs.

Indexing Japanese Content in WebHelp Pages

To optimize the indexing of Japanese content in WebHelp pages, the Kuromoji analyzer can be used. This analyzer is not included in the Oxygen XML Editor installation kit and must be downloaded and added.

To use the Kuromoji analyzer to index Japanese content in your WebHelp system, follow these steps:

1. Download the analyzer jar file from
<http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0>.
2. Place the Kuromoji analyzer jar file in the following directory: [OXYGEN INSTALLATION DIRECTORY]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.webhelp/lib.
3. For the analyzer to work properly, search terms that are entered into your WebHelp pages must be separated by spaces.

Optionally a Japanese user dictionary can be set with [*the webhelp.search.japanese.dictionary parameter*](#).

Compiled HTML Help (CHM) Output Format

To perform a **Compiled HTML Help (CHM)** transformation Oxygen XML Editor needs Microsoft HTML Help Workshop to be installed on your computer. Oxygen XML Editor automatically detects HTML Help Workshop and uses it.

 **Note:** HTML Help Workshop might fail if the files used for transformation contain accents in their names, due to different encodings used when writing the `.hhp` and `.hhc` files. If the transformation fails to produce the CHM output but the `.hhp` (HTML Help Project) file is already generated, you can manually try to build the CHM output using HTML Help Workshop.

Changing the Output Encoding

Oxygen XML Editor uses the `htmlhelp.locale` parameter to correctly display specific characters of different languages in the output of the **Compiled HTML Help (CHM)** transformation. The **Compiled HTML Help (CHM)** default scenario that comes bundled with Oxygen XML Editor has the `htmlhelp.locale` parameter set to `en-US`.

The default value of the `htmlhelp.locale` is `en-US`. To customize this parameter, go to  **Configure Transformation Scenarios** and click the  **Edit** button. In the parameter tab search for the `htmlhelp.locale` parameter and change its value to the desired language tag.

The format of the `htmlhelp.locale` parameter is `LL-CC`, where `LL` represents the language code (`en` for example) and `CC` represents the country code (`US` for example). The language codes are contained in the ISO 639-1 standard and the country codes are contained in the ISO 3166-1 standard. For further details about language tags, go to <http://www.rfc-editor.org/rfc/rfc5646.txt>.

Kindle Output Format

Oxygen XML Editor requires KindleGento generate Kindle output from DITA Maps. To install KindleGen for use by Oxygen XML Editor, follow these steps:

1. Go to www.amazon.com/kindleformat/kindlegen and download the zip file that matches your operating system.
2. Unzip the file on your local disk.
3. Start Oxygen XML Editor and open a DITA Map in the **DITA Maps Manager** view.
4. In the **DITA Maps Manager View** open the  **Configure Transformation Scenario(s)** dialog box.
5. Select the **DITA Map Kindle** transformation and press the **Edit** button to edit it.
6. Go to **Parameters** tab and set the **kindlegen.executable** parameter as the path to the KindleGen directory.
7. Accept the changes.

Migrating OOXML Documents to DITA

Oxygen XML Editor integrates the entire DITA for Publishers plugins suite, enabling you to migrate content from Open Office XML documents to DITA:

- Open an OOXML document in Oxygen XML Editor. The document is opened in the **Archive Browser** view.
- From the **Archive Browser**, open `document.xml`.
 -  **Note:** `document.xml` holds the content of the document.
- Click  **Configure Transformation Scenario(s)** on the toolbar and apply the **DOCX DITA** scenario. If you encounter any issues with the transformation, click the link below for further details about the Word to DITA Transformation Framework.

DITA Map Templates

The default templates available for DITA maps are stored in `[OXYGEN_DIR]/frameworks/dita/templates/map` folder.

Here are some of the DITA Map templates available when creating [new documents from templates](#):

- **DITA Map - Bookmap** - New DITA Bookmap.
- **DITA Map - Map** - New DITA Map.
- **DITA Map - Learning Map** - New DITA learning and training content specialization map.
- **DITA Map - Learning Bookmap** - New DITA learning and training content specialization bookmap.
- **DITA Map - Eclipse Map** - IBM specialization of DITA Map used for producing Eclipse Help plugins.

DITA for Publishers Map specialization templates:

- **D4P Map** - New DITA for Publishers Map.
- **D4P Pub-component-map** - New DITA for Publishers pub-component-map.
- **D4P Pubmap** - New DITA for Publishers pubmap.

The XHTML Document Type

The Extensible HyperText Markup Language (XHTML), is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is `html`.

The default schemas used for these documents are stored in the following locations:

- XHTML 1.0 - `[OXYGEN_DIR]/frameworks/xhtml/dtd/` or `[OXYGEN_DIR]/frameworks/xhtml/nvdl/`.
- XHTML 1.1 - `[OXYGEN_DIR]/frameworks/xhtml11/dtd/` or `[OXYGEN_DIR]/frameworks/xhtml11/schema/`.
- XHTML 5 - `[OXYGEN_DIR]/frameworks/xhtml/xhtml15 (epub3)/`.

The CSS options for the XHTML document type are set to merge the CSS stylesheets specified in the document with the CSS stylesheets defined in the XHTML document type.

The default CSS files used for rendering XHTML content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/xhtml/css/`.

The default catalogs for the XHTML document type are as follows:

- `[OXYGEN_DIR]/frameworks/xhtml/dtd/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/frameworks/relaxng/catalog.xml`
- `[OXYGEN_DIR]/frameworks/nvdl/catalog.xml`
- `[OXYGEN_DIR]/frameworks/xhtml11/dtd/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/frameworks/xhtml11/schema/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/xhtml15 (epub3)/catalog-compat.xml`

XHTML Author Actions

A variety of actions are available in the XHTML framework that can be added to the **XHTML** menu, the **Author custom actions** toolbar, the contextual menu, and the **Content Completion Assistant**. The following default actions are included in the toolbar and the **XHTML** menu and are readily available when editing in **Author** mode (most of them are also available, by default, in the contextual menu):

B Bold

Changes the style of the selected text to `bold` by surrounding it with `b` tag. You can use this action on multiple non-contiguous selections.

I Italic

Changes the style of the selected text to `italic` by surrounding it with `i` tag. You can use this action on multiple non-contiguous selections.

U Underline

Changes the style of the selected text to `underline` by surrounding it with `u` tag. You can use this action on multiple non-contiguous selections.

Insert a hypertext link

Inserts an `a` element with an `href` attribute at the caret position. You can type the URL of the reference you want to insert or use the  **Browse** drop-down list to select it using one of the following options:

- **Browse for local file** - Displays the **Open** dialog box to select a local file.
- **Browse for remote file** - Displays the **Open URL** dialog box to select a remote file.
- **Browse for archived file** - Opens the **Archive Browser** to select a file from an archive.
- **Browse Data Source Explorer** - Open the **Data Source Explorer** to select a file from a connected data source.
- **Search for file** - Opens the **Find Resource** dialog box to search for a file.

Insert image reference

Inserts a graphic object at the caret position. This is done by inserting an `img` element regardless of the current context. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG.

H Headings

A drop-down list that includes actions for inserting `h1`, `h2`, `h3`, `h4`, `h5`, `h6` elements.

 **Insert a new paragraph**

Insert a new paragraph at current cursor position.

 **Insert a MathML equation**

Opens the **XML Fragment Editor** that allows you to insert and [edit MathML notations](#).

 **Insert a step or list Item**

Inserts a new step or list item in the current list type.

 **Insert an unordered list at the caret position**

Inserts an itemized list. A child list item is also automatically inserted by default.

 **Insert an ordered list at the caret position**

Inserts an ordered list. A child list item is also automatically inserted by default.

 **Insert a definition list at the caret position**

Inserts a definition list (dl element) with one list item (a dt child element and a dd child element).

 **Sort**

Sorts a table selection.

 **Insert Table**

Opens a dialog box that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

 **Insert a new table row below the current row**

Inserts a new table row with empty cells below the current row. This action is available when the caret is positioned inside a table.

 **Insert a new table row above the current row**

Inserts a new table row with empty cells above the current row. This action is available when the caret is positioned inside a table.

 **Insert a new table column after the current column**

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

 **Insert a table cell**

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, Oxygen XML Editor a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

 **Delete a table column**

Deletes the table column located at caret position.

 **Delete a table row**

Deletes the table row located at caret position.

 **Table Join/Split Drop-Down List**

The following link actions are available from this list:

-  **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
-  **Join Cell Above** - Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.
-  **Join Cell Below** - Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.



Note: When you use **Join Cell Above** and **Join Cell Below**, Oxygen XML Editor deletes the source row if case it remains empty. The cells that span over multiple rows are also updated.

- **Split Cell To The Left** - Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column. Oxygen XML Editor decreases the column span of the source cell with one.
- **Split Cell To The Right** - Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column. Oxygen XML Editor decreases the column span of the source cell with one.
- **Split Cell Above** - Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row. Oxygen XML Editor decreases the column span of the source cell with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row. Oxygen XML Editor decreases the column span of the source cell with one.

Dragging a file from *the Project view* or *DITA Maps Manager view* and dropping it into an XHTML document that is edited in **Author** mode creates a link to the dragged file (the `a` element with the `href` attribute) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into an XHTML document inserts an image element (the `img` element with the `src` attribute) at the drop location, similar to the **Insert Image Reference** toolbar action.

XHTML Transformation Scenarios

The following default transformation scenarios are available for XHTML:

- **XHTML to DITA concept** - Converts an XHTML document to a DITA concept document.
- **XHTML to DITA reference** - Converts an XHTML document to a DITA reference document.
- **XHTML to DITA task** - Converts an XHTML document to a DITA task document.
- **XHTML to DITA topic** - Converts an XHTML document to a DITA topic document.

XHTML Templates

Default templates are available for XHTML. They are stored in `[OXYGEN_DIR]/frameworks/xhtml/templates` folder and they can be used for easily creating basic XHTML documents.

Here are some of the XHTML templates available when creating *new documents from templates*.

- **XHTML - 1.0 Strict** - New Strict XHTML 1.0
- **XHTML - 1.0 Transitional** - New Transitional XHTML 1.0
- **XHTML - 1.1 DTD Based** - New DTD based XHTML 1.1
- **XHTML - 1.1 DTD Based + MathML 2.0 + SVG 1.1** - New XHTML 1.1 with MathML and SVG insertions
- **XHTML - 1.1 Schema based** - New XHTML 1.1 XML Schema based

The TEI ODD Document Type

The **Text Encoding Initiative - One Document Does it all (TEI ODD)** is a TEI XML-conformant specification format that allows you to create a custom TEI P5 schema in a literate programming fashion. A system of XSLT stylesheets called *Roma* was created by the TEI Consortium for manipulating the ODD files.

A file is considered to be a TEI ODD document when the following conditions are true:

- The file extension is `.odd`.
- The document namespace is `http://www.tei-c.org/ns/1.0`.

The default schema, `tei_odds.rng`, used for these documents is stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/`.

The default CSS files used for rendering TEI ODD content are stored in
 [OXYGEN_DIR]/frameworks/tei/xml/tei/css/.

There are two default catalogs for the TEI ODD document type:

- [OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/catalog.xml
- [OXYGEN_DIR]/frameworks/tei/xml/tei/schema/catalog.xml

To watch our video demonstration about TEI editing, go to http://oxygentools.com/demo/WYSIWYG_TEI_Editing.html.

TEI ODD Author Actions

The following actions are available in the contextual menu, the **TEI ODD** submenu of the main menu, and in the **Author custom actions** toolbar:

B Bold

Changes the style of the selected text to **bold** by surrounding it with `hi` tag and setting the `rrend` attribute to `bold`. You can use this action on multiple non-contiguous selections.

I Italic

Changes the style of the selected text to **italic** by surrounding it with `hi` tag and setting the `rrend` attribute to `italic`. You can use this action on multiple non-contiguous selections.

U Underline

Changes the style of the selected text to **underline** by surrounding it with `hi` tag and setting the `rrend` attribute to `u1`. You can use this action on multiple non-contiguous selections.

S Insert Section

Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on;

Insert image reference

inserts an image reference at the caret position;

Insert Table

Opens a dialog box that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also automatically inserted by default.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **TEI > ID Options** list, they will all receive an unique ID

 **Note:** IDs already set are preserved.

The action is available both in the contextual menu and in the **TEI** main menu.

ID Options

Action available in the **TEI** main menu, allows you to specify the elements for which Oxygen XML Editor generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs when copying content in the same document** option is not checked.

Search References

Finds the references to the `id` attribute value of the selected element in all the topics from the current DITA map (opened in the **DITA Maps Manager** view). The default shortcut of the action is **Ctrl Shift G (Command Shift G on OS X)** and can be changed in the **DITA Topic** document type.

Dragging a file from *the Project view* or *DITA Maps Manager view* and dropping it into a TEI ODD document that is edited in **Author** mode, creates a link to the dragged file (the `ptr` element with the `target` attribute) at the drop location.

TEI ODD Transformation Scenarios

The following default transformations are available:

- **TEI ODD XHTML** - Transforms a TEI ODD document into an XHTML document
- **TEI ODD PDF** - Transforms a TEI ODD document into a PDF document using the Apache FOP engine
- **TEI ODD EPUB** - Transforms a TEI ODD document into an EPUB document
- **TEI ODD DOCX** - Transforms a TEI ODD document into a DOCX document
- **TEI ODD ODT** - Transforms a TEI ODD document into an ODT document
- **TEI ODD RelaxNG XML** - Transforms a TEI ODD document into a RelaxNG XML document
- **TEI ODD to DTD** - Transforms a TEI ODD document into a DTD document
- **TEI ODD to XML Schema** - Transforms a TEI ODD document into an XML Schema document
- **TEI ODD to RelaxNG Compact** - Transforms a TEI ODD document into an RelaxNG Compact document

TEI ODD Templates

There is only one default template which is stored in the `[OXYGEN_DIR]/frameworks/tei/templates/TEI` ODD folder and can be used for easily creating a basic TEI ODD document. This template is available when creating *new documents from templates*.

- **TEI ODD** - New TEI ODD document

The TEI P4 Document Type

The **Text Encoding Initiative (TEI) Guidelines** is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P4 document when one of the following conditions are true:

- The local name of the root is `TEI` . `2`.
- The public id of the document is `-//TEI P4`.

The default DTD schema, `tei2.dtd`, used for these documents is stored in `[OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/`.

The default CSS files used for rendering TEI P4 content in **Author** mode is stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/css/`.

The default catalogs for the TEI P4 document type are as follows:

- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/custom/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/stylesheets/catalog.xml`

To watch our video demonstration about TEI editing, go to http://oxygenvml.com/demo/WYSIWYG_TEI_Editing.html.

TEI P4 Author Actions

The following actions are available in the contextual menu, the **TEI P4** submenu of the main menu, and in the **Author custom actions** toolbar:

B Bold

Changes the style of the selected text to **bold** by surrounding it with `hi` tag and setting the `x:rend` attribute to **bold**. You can use this action on multiple non-contiguous selections.

I Italic

Changes the style of the selected text to **italic** by surrounding it with `hi` tag and setting the `x:rend` attribute to **italic**. You can use this action on multiple non-contiguous selections.

U Underline

Changes the style of the selected text to **underline** by surrounding it with `hi` tag and setting the `x:rend` attribute to `u:1`. You can use this action on multiple non-contiguous selections.

Browse reference manual

Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu) and in the documentation tip displayed by the **Content Completion Assistant**.

§ Insert Section

Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on.

🖼 Insert image reference

inserts an image reference at the caret position;

 CreateTable

Opens a dialog box that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

☰ Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also automatically inserted by default.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **TEI > ID Options** list, they will all receive an unique ID

 **Note:** IDs already set are preserved.

The action is available both in the contextual menu and in the **TEI** main menu.

ID Options

Action available in the **TEI** main menu, allows you to specify the elements for which Oxygen XML Editor generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs when copying content in the same document** option is not checked.

Search References

Finds the references to the `id` attribute value of the selected element in all the topics from the current DITA map (opened in the **DITA Maps Manager** view). The default shortcut of the action is **Ctrl Shift G (Command Shift G on OS X)** and can be changed in the **DITA Topic** document type.

Also, if you drag and drop a file from *the Project view* or *DITA Maps Manager view* into a TEI P4 document that is edited in **Author** mode, it will create a link to the dragged file (the `ptr` element with the `target` attribute) at the drop location.

TEI P4 Transformation Scenarios

The following default transformations are available:

- **TEI HTML** - Transforms a TEI document into a HTML document;
- **TEI P4 -> TEI P5 Conversion** - Convert a TEI P4 document into a TEI P5 document;
- **TEI PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine.

TEI P4 Templates

The default templates are stored in `[OXYGEN_DIR]/frameworks/tei/templates/TEI_P4` folder and they can be used for easily creating basic TEI P4 documents. These templates are available when creating [new documents from templates](#).

- **TEI P4 - Lite** - New TEI P4 Lite
- **TEI P4 - New Document** - New TEI P4 standard document

Customization of TEI Frameworks Using the Latest Sources

The **TEI P4** and **TEI P5** frameworks are available as a public project at the following SVN repository:

```
https://oxygen-tei.googlecode.com/svn/trunk/
```

This project is the base for customizing a TEI framework.

1. Check out the project on a local computer from the SVN repository.
This action is done with an SVN client application that creates a working copy of the SVN repository on a local computer.
2. Customize the TEI framework in Oxygen XML Editor.
 - a) Set the Oxygen XML Editor `frameworks` folder to the `oxygen/frameworks` subfolder of the folder of the SVN working copy.
Open the Preferences dialog box, go to **Global**, and set the path of the SVN working copy in the **Use custom frameworks** option.
 - b) *Open the Preferences dialog box*, go to **Document Type Association > Locations**, and select **Custom**.
3. Build a jar file with the TEI framework.

The SVN project includes a `build.xml` file that can be used for building a jar file using the Ant tool. The command that should be used:

```
ant -f build.xml
```

4. Distribute the jar file to the users that need the customized TEI framework.
The command from the above step creates a file `tei.zip` in the `dist` subfolder of the SVN project. Each user that needs the customized TEI framework will receive the `tei.zip` file and will unzip it in the `frameworks` folder of the Oxygen XML Editor install folder.

The TEI P5 Document Type

The **Text Encoding Initiative (TEI) Guidelines** is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P5 document when one of the following conditions are true:

- The document namespace is `http://www.tei-c.org/ns/1.0`.
- The public id of the document is `-//TEI P5`.

The default schemas used for these documents are stored in

```
[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/dtd/ or  
[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/.
```

The CSS file used for rendering TEI P5 content is located in

[OXYGEN_DIR]/frameworks/tei/xml/tei/css/tei_oxygen.css.

The default catalogs for the TEI P5 document type are as follows:

- [OXYGEN_DIR]/frameworks/tei/xml/tei/schema/dtd/catalog.xml
- [OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/dtd/catalog.xml
- [OXYGEN_DIR]/frameworks/tei/xml/tei/stylesheets/catalog.xml

To watch our video demonstration about TEI editing, go to http://oxygentools.com/demo/WYSIWYG_TEI_Editing.html.

TEI P5 Author Actions

The following actions are available in the contextual menu, the **TEI P5** submenu of the main menu, and in the **Author custom actions** toolbar:

B Bold

Changes the style of the selected text to **bold** by surrounding it with `hi` tag and setting the `r:rend` attribute to `bold`. You can use this action on multiple non-contiguous selections.

I Italic

Changes the style of the selected text to **italic** by surrounding it with `hi` tag and setting the `r:rend` attribute to `italic`. You can use this action on multiple non-contiguous selections.

U Underline

Changes the style of the selected text to **underline** by surrounding it with `hi` tag and setting the `r:rend` attribute to `u:1`. You can use this action on multiple non-contiguous selections.

Browse reference manual

Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu) and in the documentation tip displayed by the **Content Completion Assistant**.

S Insert Section

Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on.

Insert image reference

inserts an image reference at the caret position;

Insert Table

Opens a dialog box that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also automatically inserted by default.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **TEI > ID Options** list, they will all receive an unique ID

 **Note:** IDs already set are preserved.

The action is available both in the contextual menu and in the **TEI** main menu.

ID Options

Action available in the **TEI** main menu, allows you to specify the elements for which Oxygen XML Editor generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs when copying content in the same document** option is not checked.

Search References

Finds the references to the `id` attribute value of the selected element in all the topics from the current DITA map (opened in the **DITA Maps Manager** view). The default shortcut of the action is **Ctrl Shift G (Command Shift G on OS X)** and can be changed in the **DITA Topic** document type.

Also, if you drag and drop a file from *the Project view* or *DITA Maps Manager view* into a TEI P5 document that is edited in **Author** mode, it will create a link to the dragged file (the `ptr` element with the `target` attribute) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into a TEI P5 document inserts a graphic element (the `graphic` element with the `url` attribute) at the drop location, similar to the  **Insert Image Reference** toolbar action.

TEI P5 Transformation Scenarios

The following default transformations are available:

- **TEI P5 XHTML** - transforms a TEI P5 document into a XHTML document;
- **TEI P5 PDF** - transforms a TEI P5 document into a PDF document using the Apache FOP engine;
- **TEI EPUB** - transforms a TEI P5 document into an EPUB output. The EPUB output will contain any images referenced in the TEI XML document;
- **TEI DOCX** - transforms a TEI P5 document into a DOCX (OOXML) document. The DOCX document will contain any images referenced in the TEI XML document;
- **TEI ODT** - transforms a TEI P5 document into an ODT (ODF) document. The ODT document will contain any images referenced in the TEI XML document.

TEI P5 Templates

The default templates are stored in `[OXYGEN_DIR]/frameworks/tei/templates/TEI_P5` folder and they can be used for easily creating basic TEI P5 documents. These templates are available when creating *new documents from templates*:

- **TEI P5 - All** - New TEI P5 All;
- **TEI P5 - Bare** - New TEI P5 Bare;
- **TEI P5 - Lite** - New TEI P5 Lite;
- **TEI P5 - Math** - New TEI P5 Math;
- **TEI P5 - Speech** - New TEI P5 Speech;
- **TEI P5 - SVG** - New TEI P5 with SVG extensions;
- **TEI P5 - XInclude** - New TEI P5 XInclude aware.

Customization of TEI Frameworks Using the Latest Sources

The **TEI P4** and **TEI P5** frameworks are available as a public project at the following SVN repository:

<https://oxygen-tei.googlecode.com/svn/trunk/>

This project is the base for customizing a TEI framework.

1. Check out the project on a local computer from the SVN repository.

This action is done with an SVN client application that creates a working copy of the SVN repository on a local computer.

2. Customize the TEI framework in Oxygen XML Editor.

- a) Set the Oxygen XML Editor frameworks folder to the oxygen/frameworks subfolder of the folder of the SVN working copy.
Open the Preferences dialog box, go to **Global**, and set the path of the SVN working copy in the **Use custom frameworks** option.
- b) *Open the Preferences dialog box*, go to **Document Type Association > Locations**, and select **Custom**.

3. Build a jar file with the TEI framework.

The SVN project includes a build.xml file that can be used for building a jar file using the Ant tool. The command that should be used:

```
ant -f build.xml
```

4. Distribute the jar file to the users that need the customized TEI framework.

The command from the above step creates a file tei.zip in the dist subfolder of the SVN project. Each user that needs the customized TEI framework will receive the tei.zip file and will unzip it in the frameworks folder of the Oxygen XML Editor install folder.

Customization of TEI Frameworks Using the Compiled Sources

The following procedure describes how to update to the latest stable version of TEI Schema and TEI XSL, already integrated in the TEI framework for Oxygen XML Editor.

1. Go to <https://code.google.com/p/oxygen-tei/>;
2. Go to **Downloads**;
3. Download the latest uploaded .zip file;
4. Unpack the .zip file and copy its content in the Oxygen XML Editor frameworks folder.

The JATS Document Type

The JATS (NISO Journal Article Tag Suite) document type is a technical standard that defines an XML format for scientific literature.

A file is considered to be a JATS document when the PUBLIC ID of the document contains the string -//NLM//DTD.

The default schemas for the JATS document types are stored in [OXYGEN_DIR]/frameworks/jats/O2-DTD/.

The default CSS files used for rendering JATS content in **Author** mode are stored in [OXYGEN_DIR]/frameworks/jats/css/.

The default XML catalog, JATS-catalog-O2.xml, is stored in [OXYGEN_DIR]/frameworks/O2-DTD/.

JATS Author Actions

A variety of actions are available in the JATS framework that can be added to the **JATS** menu, the **Author custom actions** toolbar, the contextual menu, and the **Content Completion Assistant**. The following default actions are included in the toolbar, contextual menu, and the **JATS** menu and are readily available when editing in **Author** mode:

B Bold

Surrounds the selected text with a bold tag. You can use this action on multiple non-contiguous selections.

I Italic

Surrounds the selected text with an italic tag. You can use this action on multiple non-contiguous selections.

U Underline

Surrounds the selected text with an underline tag. You can use this action on multiple non-contiguous selections.

P Insert a new paragraph

Insert a new paragraph at current cursor position.

Insert image reference

Inserts an image reference at the caret position. Depending on the current context, an image-type element is inserted.

Insert a step or list Item

Inserts a new step or list item in the current list type.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also automatically inserted by default.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also automatically inserted by default.

Drag/Drop Actions

Dragging a file from *the Project view* or *DITA Maps Manager view* and dropping it into a JATS document that is edited in **Author** mode, creates a link to the dragged file (the `ext-link` element with the `xlink:href` attribute) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into a JATS document inserts an image element (the `inline-graphic` element with the `xlink:href` attribute) at the drop location, similar to the **Insert Image Reference** toolbar action.

JATS Transformation Scenarios

The following default transformation scenario is available for JATS documents:

- **JATS Preview (simple HTML)** - Converts a JATS document to a simple HTML document.

JATS Templates

Default templates are available for JATS documents. They are stored in `[OXYGEN_DIR]/frameworks/jats/templates` folder and they can be used for easily creating basic JATS documents.

The default JATS templates that are available when creating *new documents from templates* are as follows:

- **Archiving** - JATS archiving tag set version 1.0.
- **Authoring** - JATS authoring tag set version 1.0.
- **Book** - JATS book tag set version 1.0.
- **Publishing** - JATS publishing tag set version 1.0.

The EPUB Document Type

Three distinct frameworks are supported for the EPUB document type:

- **NCX** - A declarative global navigation definition.
- **OCF** - The Open Container Format (OCF) defines a mechanism by which all components of an Open Publication Structure (OPS) can be combined into a single file system entity.
- **OPF** - The Open Packaging Format (OPF) defines the mechanism by which all components of a published work that conforms to the Open Publication Structure (OPS) standard (including metadata, reading order, and navigational information) are packaged in an OPS Publication.



Note: Oxygen XML Editor supports both OPF 2.0 and OPF 3.0.

Document Templates

The default templates for the *NCX* and *OCF* document types are located in the `[OXYGEN_DIR]/frameworks/docbook/templates` folder.

The default template for the *OPF 2.0* document type is located in the `[OXYGEN_DIR]/frameworks/docbook/templates/2.0` folder.

The default template for the *OPF 3.0* document type is located in the [OXYGEN_DIR]/frameworks/docbook/templates/3.0 folder.

The following EPUB templates are available when creating *new documents from templates*:

- **NCX - Toc** - New table of contents.
- **OCF - Container** - New container based OCF.
- **OCF - Encryption** - New encryption based OCF.
- **OCF - Signatures** - New signature based OCF.
- **OPF 2.0 - Content (2.0)** - New OPF 2.0 content.
- **OPF 3.0 - Content (3.0)** - New OPF 3.0 content.

The DocBook Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with the DocBook olink.

A file is considered to be a *Targetset* when the root name is targetset.

The default schema, targetdatabase.dtd, for this type of document is stored in [OXYGEN_DIR]/frameworks/docbook/xsl/common/.

Document Templates

The default template for DocBook *Targetset* documents is located in the [OXYGEN_DIR]/frameworks/docbook/templates/Targetset folder.

The following DocBook *Targetset* template is available when creating *new documents from templates*:

- **DocBook Targetset - Map** - New Targetset map.

Chapter

10

Authoring Customization

Topics:

- *Authoring Customization Guide*
- *API Frequently Asked Questions (API FAQ)*

This section contains an *Authoring Customization Guide* and a collection of *Frequently Asked Questions regarding the Oxygen XML Editor API*.

Authoring Customization Guide

The **Author** mode editor of Oxygen XML Editor was designed to provide a friendly user-interface for editing XML documents. **Author** combines the power of source editing with the intuitive interface of a word processor. You can customize the **Author** mode editor to support new custom XML formats or to change how standard XML formats are edited.



Figure 244: Oxygen XML Editor Author Visual Editor

Although Oxygen XML Editor comes with already configured frameworks for DocBook, DITA, TEI, and XHTML you might need to create a customization of the editor to handle other types of documents. A common use case is when your organization holds a collection of XML document types used to define the structure of internal documents and they need to be visually edited by people with no experience working with XML files.

There are several ways to customize the editor:

1. Create a CSS file defining styles for the XML elements the user will work with, and create XML files that reference the CSS through an `<xml-stylesheet>` processing instruction.
2. Fully configure a document type association. This involves putting together the CSS stylesheets, XML schemas, actions, menus, bundling them, and distributing an archive. The CSS and GUI elements are settings for the Oxygen XML Editor **Author** mode. The other settings such as the templates, catalogs, and transformation scenarios are general settings and are enabled whenever the association is active, regardless of the editing mode (**Text**, **Grid**, or **Author**).

Simple Customization Tutorial

The most important elements of a document type customization are represented by an XML Schema to define the XML structure, the CSS to render the information and the XML instance template which links the first two together.

XML Schema

Let's consider the following XML Schema, `test_report.xsd` defining a report with results of a testing session. The report consists of a title, few lines describing the test suite that was run and a list of test results, each with a name and a boolean value indicating if the test passed or failed.

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
  <xss:element name="report">
    <xss:complexType>
      <xss:sequence>
        <xss:element ref="title"/>
        <xss:element ref="description"/>
      </xss:sequence>
    </xss:complexType>
  </xss:element>
</xss:schema>
```

```

        <xs:element ref="results"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
<xs:element name="description">
    <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="line">
                <xs:complexType mixed="true">
                    <xs:sequence minOccurs="0"
                        maxOccurs="unbounded">
                        <xs:element name="important"
                            type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="results">
    <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="entry">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="test_name"
                            type="xs:string"/>
                        <xs:element name="passed"
                            type="xs:boolean"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>

```

The use-case is that several users are testing a system and must send report results to a content management system. The Author customization should provide a visual editor for this kind of documents.

CSS Stylesheet

A set of rules must be defined for describing how the XML document is to be rendered in **Author** mode. This is done using Cascading Style Sheets (CSS). CSS is a language used to describe how an HTML or XML document should be formatted by a browser. CSS is widely used in the majority of websites.

The elements from an XML document are displayed in the layout as a series of boxes. Some of the boxes contain text and may flow one after the other, from left to right. These are called in-line boxes. There are also other type of boxes that flow one below the other, like paragraphs. These are called block boxes.

For example, consider the way a traditional text editor arranges the text. A paragraph is a block, because it contains a vertical list of lines. The lines are also blocks. However, blocks that contains in-line boxes arrange its children in a horizontal flow. That is why the paragraph lines are also blocks, while the traditional "bold" and "italic" sections are represented as in-line boxes.

The CSS allows us to specify that some elements are displayed as tables. In CSS, a table is a complex structure and consists of rows and cells. The `table` element must have children that have a *table-row* style. Similarly, the `row` elements must contain elements with a *table-cell* style.

To make it easy to understand, the following section describes how each element from a schema is formatted using a CSS file. Please note that this is just one of infinite possibilities for formatting the content.

report

This element is the root element of a report document. It should be rendered as a box that contains all other elements. To achieve this the display type is set to **block**. Additionally, some margins are set for it. The CSS rule that matches this element is:

```

report{
    display:block;
    margin:1em;
}

```

title

The title of the report. Usually titles have a large font. The **block** display is used so that the subsequent elements will be placed below it, and its font is changed to double the size of the normal text.

```
title {
    display:block;
    font-size:2em;
}
```

description

This element contains several lines of text describing the report. The lines of text are displayed one below the other, so the description has the **block** display. Also, the background color is changed to make it standout.

```
description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}
```

line

A line of text in the description. A specific aspect is not defined and it just indicates that the display should be **block** style.

```
line {
    display:block;
}
```

important

The **important** element defines important text from the description. Since it can be mixed with text, its display property must be set to **inline**. Also, the text is emphasized with **bold** to make it easier to spot.

```
important {
    display:inline;
    font-weight:bold;
}
```

results

The **results** element shows the list of *test_names* and the results for each one. To make it easier to read, it is displayed as a **table**, with a green border and margins.

```
results{
    display:table;
    margin:2em;
    border:1px solid green;
}
```

entry

An item in the results element. The results are displayed as a table so the entry is a row in the table. Thus, the display is **table-row**.

```
entry {
    display:table-row;
}
```

test_name, passed

The name of the individual test, and its result. They are cells in the results table with the display set to **table-cell**. Padding and a border are added to emphasize the table grid.

```
test_name, passed{
    display:table-cell;
    border:1px solid green;
    padding:20px;
}
```

```

passed{
    font-weight:bold;
}

```

The full content of the CSS file `test_report.css` is:

```

report {
    display:block;
    margin:1em;
}

description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}

line {
    display:block;
}

important {
    display:inline;
    font-weight:bold;
}

title {
    display:block;
    font-size:2em;
}

results{
    display:table;
    margin:2em;
    border:1px solid green;
}

entry {
    display:table-row;
}

test_name, passed{
    display:table-cell;
    border:1px solid green;
    padding:20px;
}

passed{
    font-weight:bold;
}

```

report description line important

xmlstylesheet type="text/css" href="test_report.css"

Automated test report

This is the report of the test automatically ran. Each test suite is ran at 20:00h each day. Please check the failed ones!

Database connection test	true
XSLT Transformation test	true
DTD validation test	false

Text Grid Author

Figure 245: Report Rendered in Author Mode



Note: You can edit attributes in-place in the **Author** mode using *form-based controls*.

Associating a Stylesheet with an XML Document

The tagless rendering of an XML document in the **Author** mode is driven by a CSS stylesheet which conforms to the [version 2.1 of the CSS specification](#) from the W3C consortium. Some CSS 3 features, such as namespaces and custom extensions, of the CSS specification are also supported. Oxygen XML Editor also supports stylesheets coded with the LESS dynamic stylesheet language.

There are several methods for associating a stylesheet (CSS or LESS) with an XML document:

1. Insert the `<xml-stylesheet type="text/css" href="test.css">`
- If you do not want to alter your XML documents, *you should set-up a document type*.

CSS example:

```
<?xml-stylesheet type="text/css" href="test.css"?>
```

LESS example:

```
<?xml-stylesheet type="text/css" href="test.less"?>
```



Note: XHTML documents need a `link` element, with the `href` and `type` attributes in the `head` child element, as specified in the [W3C CSS specification](#). XHTML example:

```
<link href="/style/screen.css" rel="stylesheet" type="text/css"/>
```

2. Configure a *Document Type Association* by adding a new CSS or LESS file in the settings. To do so, [open the Preferences dialog box](#) and go to **Document Type Association**. Edit the appropriate framework, open the **Author** tab, then the **CSS** tab. Press the New button to add a new CSS or LESS file.



Note: The Document Type Associations are read-only, so you need to extend an existing one.

The XML Instance Template

Based on the XML Schema and CSS file Oxygen XML Editor can help the content author in loading, editing, and validating the test reports. An XML file template must be created, which is a kind of skeleton that the users can use as a starting point for creating new test reports. The template must be generic enough and reference the XML Schema file and the CSS stylesheet.

This is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="test_report.xsd">
  <title>Automated test report</title>
  <description>
    <line>This is the report of the test automatically ran. Each test suite is ran at 20:00h each day. Please <important>check</important> the failed ones!</line>
  </description>
  <results>
    <entry>
      <test_name>Database connection test</test_name>
      <passed>true</passed>
    </entry>
    <entry>
      <test_name>XSLT Transformation test</test_name>
      <passed>true</passed>
    </entry>
    <entry>
      <test_name>DTD validation test</test_name>
      <passed>false</passed>
    </entry>
  </results>
</report>
```

The processing instruction `xmlstylesheet` associates the CSS stylesheet to the XML file. The `href` pseudo attribute contains the URI reference to the stylesheet file. In our case the CSS is in the same directory as the XML file.

The next step is to place the XSD file and the CSS file on a web server and modify the template to use the HTTP URLs, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
  href="http://www.mysite.com/reports/test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
  "http://www.mysite.com/reports/test_report.xsd">
  <title>Test report title</title>
  <description>
  ....
```

The alternative is to create an archive containing the `test_report.xml`, `test_report.css` and `test_report.xsd` and send it to the content authors.

Advanced Customization Tutorial - Document Type Associations

Oxygen XML Editor supports individual document types and classes of document types through frameworks. A framework associates a document type or a class of documents with CSS stylesheets, validation schemas, catalog files, new files templates, transformation scenarios and custom actions.

In this tutorial, we create a framework for a set of documents. As an example, we create a light documentation framework (similar to DocBook), then we set up a complete customization of the **Author** mode.

You can find the samples used in this tutorial in the [Example Files Listings](#) and the complete source code in the Simple Documentation Framework project. This project is included in the [Oxygen SDK](#), available as a Maven archetype. More information about the Oxygen SDK setup can be found [here](#).

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

Document Type Settings

To add or edit a *Document Type Association*, [open the Preferences dialog box](#) and go to **Document Type Association**. All the changes can be made in the **Document Type** editing dialog box.

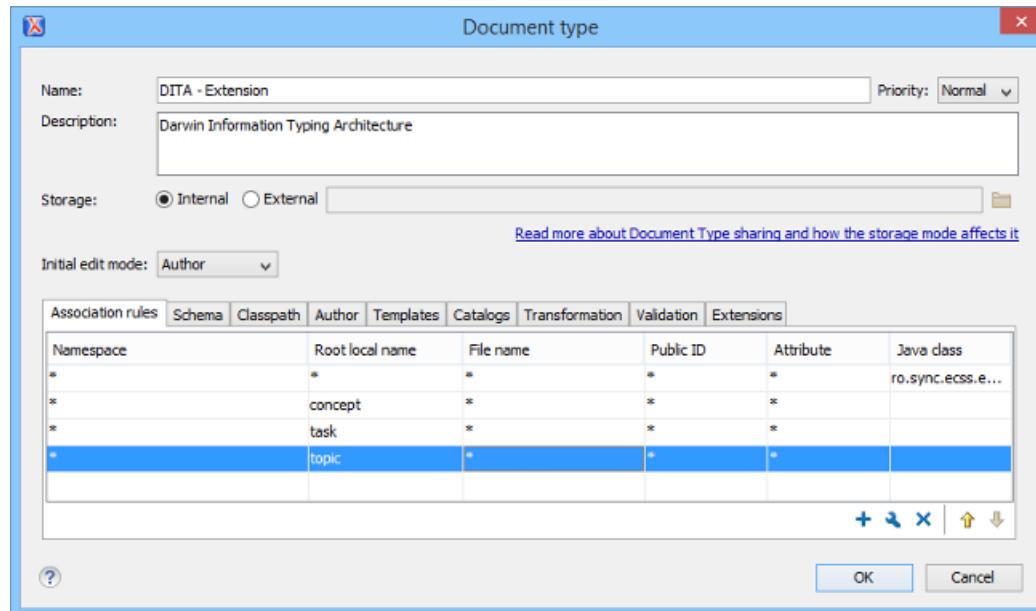


Figure 246: The Document Type Dialog Box

You can specify the following properties for a document type:

- **Name** - The name of the document type.
- **Priority** - When multiple document types match the same document, the priority determines the order in which they are applied. It can be one of the following: Lowest, Low, Normal, High, Highest. The predefined document types that are already configured when the application is installed on the computer have the default Low priority.



Note: Frameworks that have the same priority are sorted alphabetically.

- **Description** - The document type description displayed as a tool tip in the *Document Type Association table*.
- **Storage** - The location where the document type is saved. If you select the **External** storage option, the document type is saved in the specified file with a mandatory framework extension, located in a subdirectory of the current frameworks directory. If you select the **Internal** storage option, the document type data is saved in the current .xpr Oxygen XML Editor project file (for Project-level Document Type Association options) or in the Oxygen XML Editor internal options (for Global-level Document Type Association Options). You can change the Document Type Association options level in the *Document Type Association options*.
- **Initial edit mode** - Allows you to select the initial editing mode for this document type: **Editor specific**, **Text**, **Author**, **Grid** and **Design** (available only for the W3C XML Schema editor). If the **Editor specific** option is selected, the initial editing mode is determined based upon the editor type. You can find the mapping between editors and edit modes in the *Edit modes preferences page*. You can impose an initial mode for opening files that match the association rules of the document type. For example, if the files are usually edited in the **Author** mode you can set it in the **Initial edit mode** combo box.



Note: You can also customize the initial mode for a document type in the **Edit modes** preferences page. *Open the Preferences dialog box* and go to **Editor > Edit modes**.

You can specify the **Association rules** used for determining a document type for an opened XML document. A rule can define one or more conditions. All conditions need to be fulfilled in order for a specific rule to be chosen. Conditions can specify:

- **Namespace** - The namespace of the document that matches the document type.
- **Root local name of document** - The local name of the document that matches the document type.
- **File name** - The file name (including the extension) of the document that matches the document type.
- **Public ID** (for DTDs) - The PUBLIC identifier of the document that matches the document type.
- **Attribute** - This field allows you to associate a document type depending on a certain value of the attribute in the root.
- **Java class** - Name of the Java class that is called to determine if the document type should be used for an XML document. Java class must implement `ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher` interface from *Author API*.

In the **Schema** tab, you can specify the type and URI of schema used for validation and content completion of all documents from the document type, when there is no schema detected in the document.

You can choose one of the following schema types:

- DTD
- Relax NG schema (XML syntax)
- Relax NG schema (XML syntax) + Schematron
- Relax NG schema (compact syntax)
- XML Schema
- XML Schema + Schematron rules
- NVDL schema

Configure Actions, Menus, and Toolbars for a Framework

You can configure actions, menus, and toolbars that are specific to a document type in the **Author** mode to gain a productive editing experience, by using the **Document Type** dialog box.

To add or configure actions, menus, or toolbars follow this procedure:

1. Open the [Preferences dialog box](#), go to **Document Types Association**, and click the framework for which you want to create an action.
2. Click **Edit** and in the **Document Type** dialog box go to the **Author** tab, then go to **Actions**.
3. Click the **New** button and use the [Action dialog box](#) to create an action.

Configure the Insert Section Action for a Framework

This section presents all the steps that you need to follow, to define the **Insert Section** action. It is assumed that the icon files, (Section16.gif) for the menu item and (Section20.gif) for the toolbar, are already available. Although you could use the same icon size for both the menu and toolbar, usually the icons from the toolbars are larger than the ones found in the menus. These files should be placed in the frameworks/sdf directory.

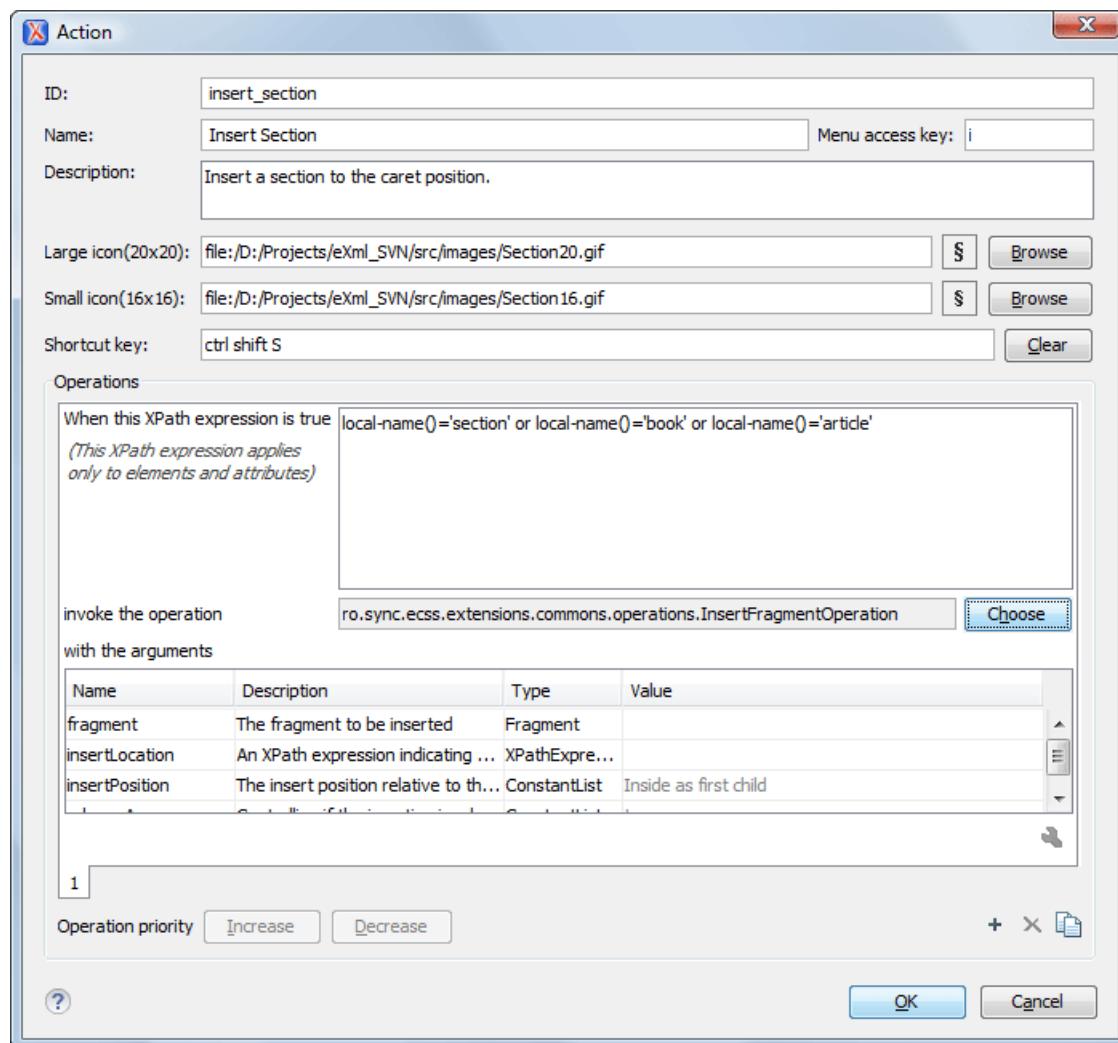


Figure 247: The Action Dialog Box

1. Set the **ID** field to **insert_section**. This is an unique action identifier.
2. Set the **Name** field to **Insert Section**. This will be the action's name, displayed as a tooltip when the action is placed in the toolbar, or as the menu item name.
3. Set the **Menu access key** to **i**. On Windows, the menu items can be accessed using **ALT+letter** keys combination, when the menu is visible. The letter is visually represented by underlining the first letter from the menu item name having the same value.
4. Set the **Description** field to **Insert a section at caret position**.

5. Set the **Large icon (20x20)** field to \${frameworks}/sdf/Section20.gif. A good practice is to store the image files inside the framework directory and use *editor variable* \${framework} to make the image relative to the framework location.

If the images are bundled in a jar archive together with some Java operations implementation for instance, it might be convenient for you to reference the images not by the file name, but by their relative path location in the class-path.

If the image file Section20.gif is located in the **images** directory inside the jar archive, you can reference it by using /images/Section20.gif. The jar file must be added into the **Classpath** list.

6. Set the **Small icon (16x16)** field to \${frameworks}/sdf/Section16.gif.
7. Click the text field next to **Shortcut key** and set it to **Ctrl (Meta on Mac OS)+Shift+S**. This will be the key combination to trigger the action using the keyboard only.

The shortcut is enabled only by *adding the action to the main menu of the Author mode* which contains all the actions that the author will have in a menu for the current document type.

8. At this time the action has no functionality added to it. Next you must define how this action operates. An action can have multiple operation modes, each of them activated by the evaluation of an XPath version 2.0 expression. The first enabled action mode will be executed when the action is triggered by the user. The scope of the XPath expression must be only element nodes and attribute nodes of the edited document, otherwise the expression will not return a match and will not fire the action. For this example we'll suppose you want allow the action to add a section only if the current element is either a book, article or another section.

- a) Set the XPath expression field to:

```
local-name()='section' or local-name()='book' or  
local-name()='article'
```

- b) Set the **invoke operation** field to **InsertFragmentOperation** built-in operation, designed to insert an XML fragment at caret position. This belongs to a set of built-in operations, a complete list of which can be found in the *Author Default Operations* section. This set can be expanded with your own Java operation implementations.
- c) Configure the arguments section as follows:

```
<section xmlns=  
"http://www.oxygenxml.com/sample/documentation">  
    <title/>  
</section>
```

insertLocation - leave it empty. This means the location will be at the caret position.

insertPosition - select "**Inside**".

Configure the Insert Table Action for a Framework

The procedure described below will create an action that inserts a table with three rows and three columns into a document. The first row is the table header. As with *the insert section action*, you will use the **InsertFragmentOperation** built-in operation.

Place the icon files for the menu item, and for the toolbar, in the frameworks/sdf directory.

1. Set **ID** field to **insert_table**.
2. Set **Name** field to **Insert table**.
3. Set **Menu access key** field to **t**.
4. Set **Description** field to **Adds a section element**.
5. Set **Toolbar icon** to \${framework} / toolbarIcon.png.
6. Set **Menu icon** to \${framework} / menuIcon.png.
7. Set **Shortcut key** to **Ctrl Shift T (Command Shift T on OS X)**.
8. Set up the action's functionality:
 - a) Set **XPath expression** field to **true()**.

true() is equivalent with leaving this field empty.

- b) Set **Invoke operation** to use **InvokeFragmentOperation** built-in operation that inserts an XML fragment to the caret position.
- c) Configure operation's arguments as follows:

fragment - set it to:

```
<table xmlns=
"http://www.oxygenxml.com/sample/documentation">
<header><td><td><td></header>
<tr><td><td><td></tr>
<tr><td><td><td></tr>
</table>
```

insertLocation - to add tables at the end of the section use the following code:

```
ancestor::section/*[last()]
```

insertPosition - Select **After**.

Configure the Main Menu for a Framework

Defined actions can be grouped into customized menus in the Oxygen XML Editor menu bar.

1. Open the **Document Type** dialog box for the **SDF framework** and click on the **Author** tab.
2. Click on the **Menu** label. In the left side you have the list of actions and some special entries:
 - **Submenu** - Creates a submenu. You can nest an unlimited number of menus.
 - **Separator** - Creates a separator into a menu. This way you can logically separate the menu entries.
3. The right side of the panel displays the menu tree with **Menu** entry as root. To change its name click on this label to select it, then press the **Edit** button. Enter **SD Framework** as name, and **D** as menu access key.
4. Select the **Submenu** label in the left panel section and the **SD Framework** label in the right panel section, then press the **Add as child** button. Change the submenu name to **Table**, using the **Edit** button.
5. Select the **Insert section** action in the left panel section and the **Table** label in the right panel section, then press the **Add as sibling** button.
6. Now select the **Insert table** action in the left panel section and the **Table** in the right panel section. Press the **Add as child** button.

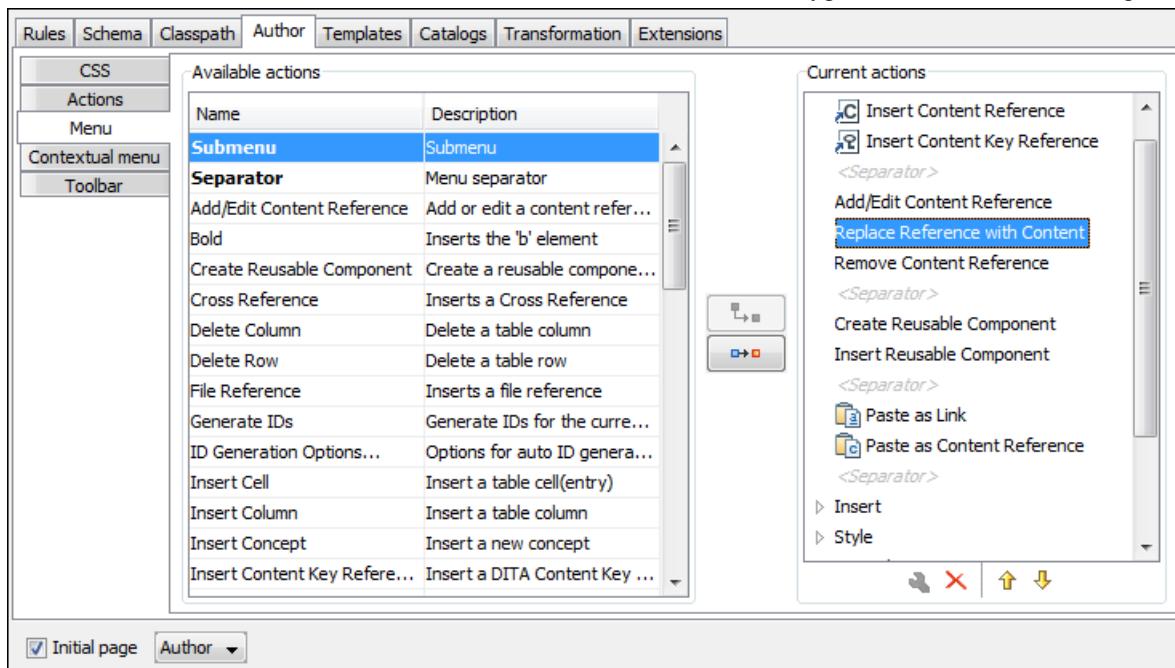


Figure 248: Configuring the Menu

When opening a **Simple Documentation Framework** test document in Author mode, the menu you created is displayed in the editor menu bar, between the **Tools** and the **Document** menus. The upper part of the menu contains generic Author actions (common to all document types) and the two actions created previously (with **Insert table** under the **Table** submenu).

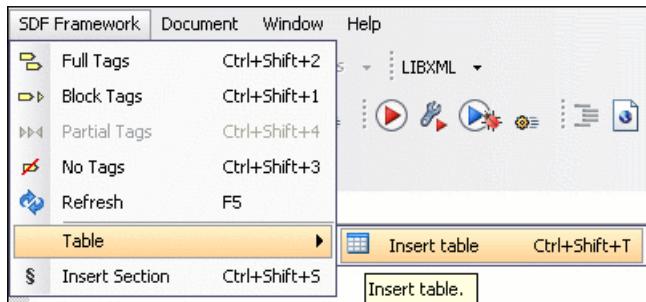


Figure 249: Author Menu

Configure the Contextual Menu for a Framework

The contextual menu is displayed when you right-click (**Ctrl** (**Meta** on Mac OS) + mouse click on Mac) in the Author editing area. You can only configure the bottom part of the menu, since the top part is reserved for a list of generic actions (such as Copy, Paste, Undo, etc.)

1. Open the **Document Type** dialog box for the particular framework and click on the **Author** tab. Next, click on the **Contextual Menu** subtab.
2. Follow the same steps as explained in the [Configuring the Main Menu](#), except changing the menu name because the contextual menu does not have a name.



Note: You can choose to reuse a submenu that contains general authoring actions. In this case, all actions (both general and document type-specific ones) are grouped together under the same submenu.

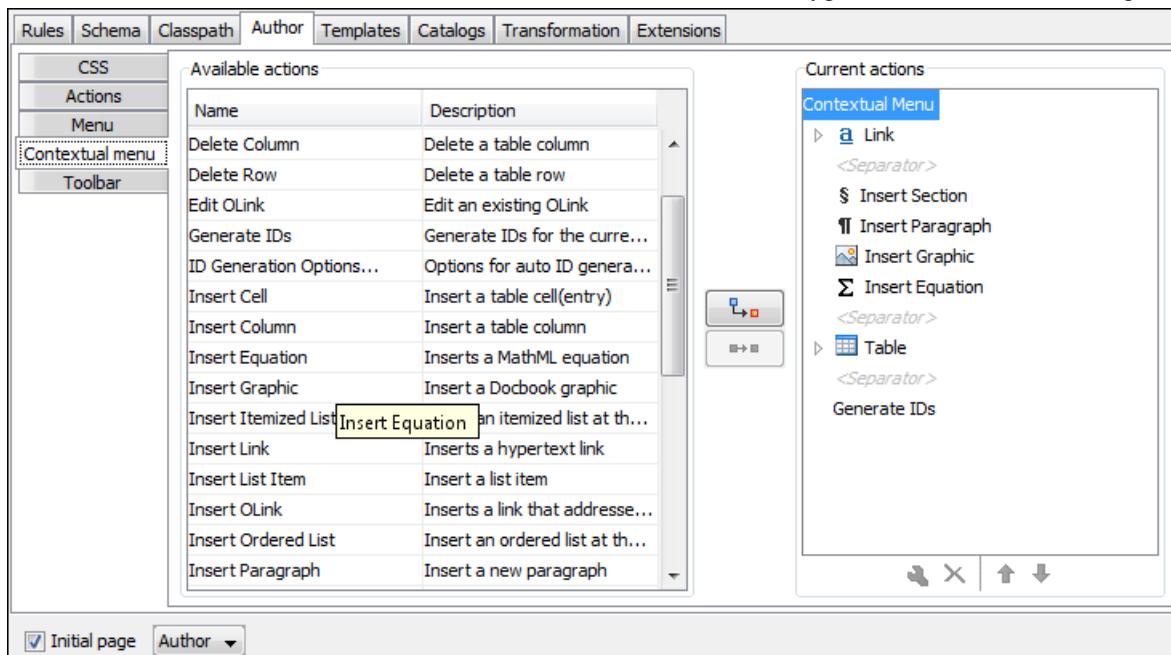


Figure 250: Configuring the Contextual Menu

To test it, open the test file, and open the contextual menu. In the lower part there is shown the **Table** sub-menu and the **Insert section** action.

Configure the Toolbars for a Framework

The procedure below describes how to add defined actions to a toolbar. These steps use examples from the two previous help topics that described how to define the **Insert Section** and **Insert Table** actions. You can also configure additional toolbars to add other custom actions.

1. Open the **Document Type** dialog box for the **SDF framework** and select the **Author** tab. Next click on the **Toolbar** label.

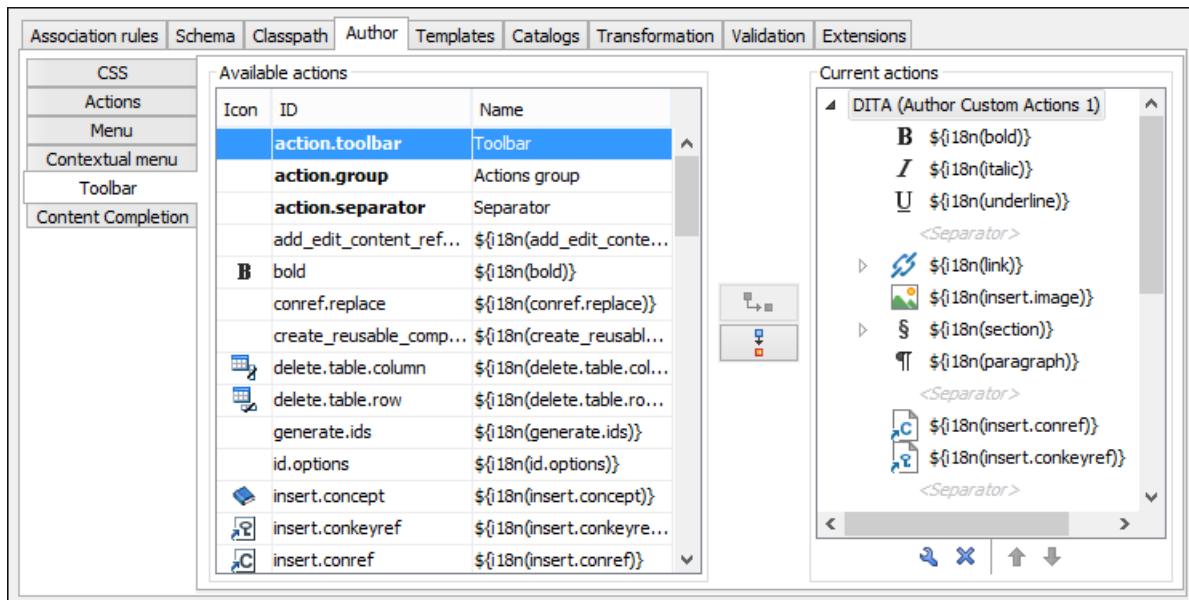


Figure 251: Configuring the Toolbar

The panel is divided in two sections: the left side contains a list of actions, while the right one contains an action tree, displaying the list of actions added in the toolbar. The special entry called *Separator* allows you to visually separate the actions in the toolbar.

2. Select the **Insert section** action in the left panel section and the **Toolbar** label in the right panel section, then press the **Add as child** button.
3. Select the **Insert table** action in the left panel section and the **Insert section** in the right panel section. Press the **Add as sibling** button.
4. When opening a **Simple Documentation Framework** test document in **Author** mode, the toolbar below will be displayed at the top of the editor.

Figure 252: Author Custom Actions Toolbar



Tip: If you have many custom toolbar actions, or want to group actions according to their category, add additional toolbars with custom names and split the actions to better suit your purpose. In case your toolbar is not displayed when switching to the **Author** mode, right click the main toolbar and make sure the entry labeled **Author custom actions 1** is enabled.

Configure Content Completion for a Framework

You can customize the content of the following **Author** controls, adding items (which, when invoked, perform custom actions) or filtering the default contributed ones:

- **Content Completion** window
- **Elements** view
- **Element Insert** menus (from the **Outline** view or breadcrumb contextual menus)

You can use the content completion customization support in the *Simple Documentation Framework* following the next steps:

1. Open the **Document type** dialog box for the **SDF framework** and select the **Author** tab. Next click on the **Content Completion** tab.

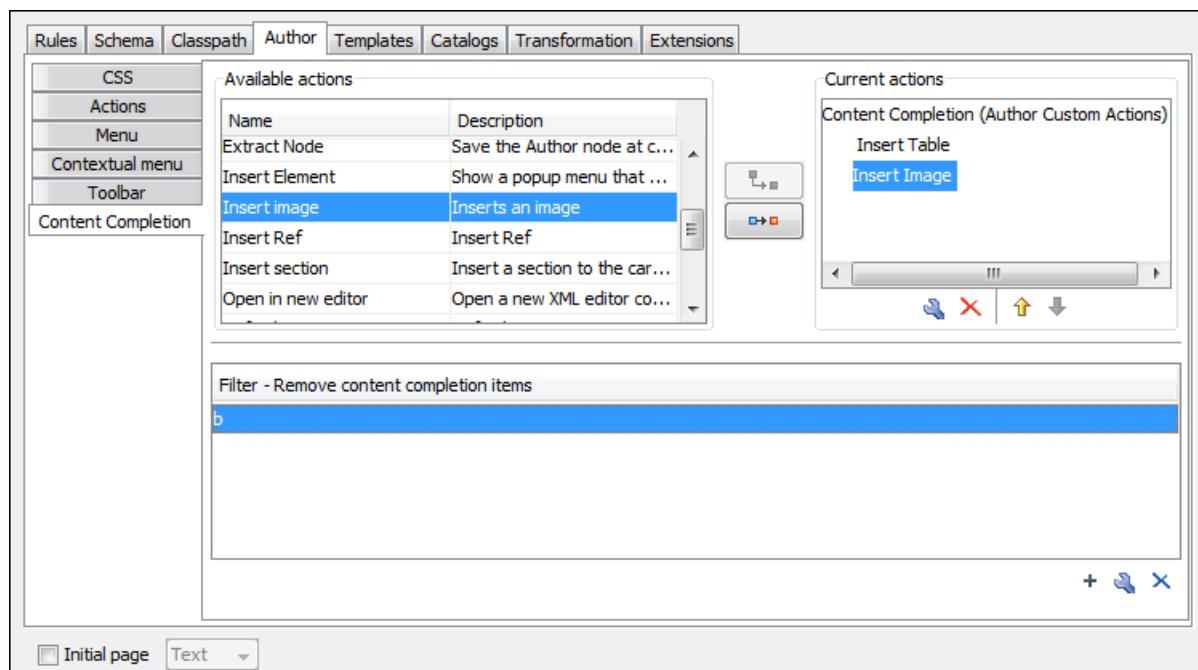


Figure 253: Customize Content Completion

The top side of the **Content Completion** section contains the list with all the actions defined within the simple documentation framework and the list of actions that you decided to include in the **Content Completion Assistant** list of proposals. The bottom side contains the list with all the items that you decided to remove from the **Content Completion Assistant** list of proposals.

2. If you want to add a custom action to the list of current **Content Completion** items, select the action item from the **Available actions** list and press the **Add as child** or **Add as sibling** button to include it in the **Current actions** list. An **Insert Action** dialog box appears, giving you the possibility to select where to provide the selected action.

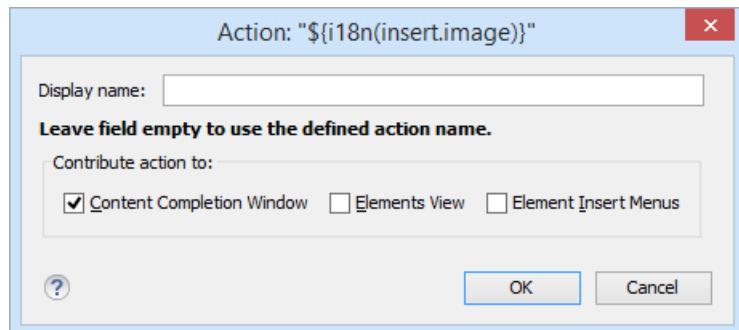


Figure 254: Insert Action Dialog Box

3. If you want to exclude a certain item from the **Content Completion** items list, you can use the **Add** button from the **Filter - Remove content completion items** list. The **Remove item** dialog box is displayed, allowing you to input the item name and to choose the controls that filter it. The **Item name** combo box accepts wildcards.

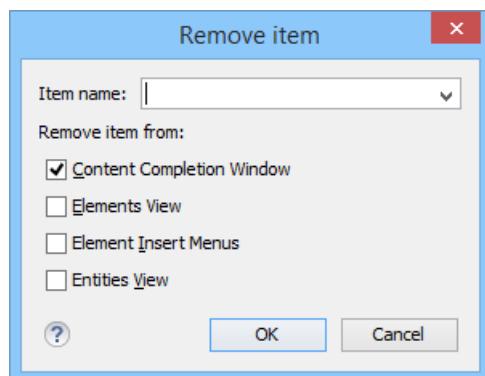


Figure 255: Remove Item Dialog Box

Author Mode Default Operations

The default operations for the **Author** mode, along with their arguments are as follows:

- **InsertFragmentOperation**

Inserts an XML fragment at the current cursor position. The selection - if there is one, remains unchanged. The fragment will be inserted in the current context of the cursor position meaning that if the current XML document uses some namespace declarations then the inserted fragment must use the same declarations. The inserted fragment will not be copied and pasted to the cursor position, but the namespace declarations of the fragment will be adapted if needed to the existing namespace declarations of the XML document. For more details about the list of parameters go to [The arguments of InsertFragmentOperation operation](#) on page 538.

- **InsertOrReplaceFragmentOperation**

Similar to **InsertFragmentOperation**, except it removes the selected content before inserting the fragment.

- **InsertOrReplaceTextOperation**

Inserts a text at current position removing the selected content, if any. The argument of this operation is:

- **text** - The text section to insert.
- **SurroundWithFragmentOperation**

Surrounds the selected content with a text fragment. Since the fragment can have multiple nodes, the surrounded content will be always placed in the first leaf element. If there is no selection, the operation will simply insert the fragment at the caret position. For more details about the list of parameters go to [The arguments of SurroundWithFragmentOperation](#) on page 539.

- **SurroundWithTextOperation**

This operation has two arguments (two text values) that will be inserted before and after the selected content. If there is no selected content, the two sections will be inserted at the caret position. The arguments of the operation are:

- **header** - The text that is placed before the selection.
- **footer** - The text that is placed after the selection.

- **InsertEquationOperation**

Inserts a fragment containing a MathML equation at caret offset. The argument of this operation is:

- **fragment** - The XML fragment containing the MathML content which should be inserted.

- **OpenInSystemAppOperation**

Opens a resource in the system application that is associated with the resource in the operating system. The arguments of this operation is:

- **resourcePath** - An XPath expression that, when executed, returns the path of the resource to be opened. Editor variables are expanded in the value of this parameter, before the expression is executed.
- **isUnparsedEntity** - Possible values are `true` or `false`. If the value is `true`, the value of the **resourcePath** argument is treated as the name of an unparsed entity.

- **InsertXIncludeOperation**

Insert an **XInclude** element at caret offset.

- **ChangeAttributeOperation**

This operation allows adding/modifying/removing an attribute. You can use this operation in your own Author action to modify the value for a certain attribute on a specific XML element. The arguments of the operation are:

- **name** - The attribute local name.
- **namespace** - The attribute namespace.
- **elementLocation** - The XPath location that identifies the element.
- **value** - The new value for the attribute. If empty or null the attribute will be removed.
- **editAttribute** - If an in-place editor exists for this attribute, it will automatically activate the in-place editor and start editing.
- **removeIfEmpty** - The possible values are `true` and `false`. True means that the attribute should be removed if an empty value is provided. The default behavior is to remove it.

- **UnwrapTagsOperation**

This operation allows removing the element tags either from the current element or for an element identified with an XPath location. The argument of the operation is

- **unwrapElementLocation** - An XPath expression indicating the element to unwrap. If it is not defined, the element at caret is unwrapped.

- **ToggleSurroundWithElementOperation**

This operation allows wrapping and unwrapping content in a specific wrapper element which can have certain attributes specified on it. It is useful to implement toggle actions such as highlighting text as bold, italic, or underline.

The operation supports processing multiple selection intervals, such as multiple cells within a table column selection.

The arguments of the operation are:

- **element** - The element to wrap or unwrap content.
- **schemaAware** - This argument applies only on the surround with element operation and controls whether or not the insertion is valid, based upon the schema. If the insertion is not valid, then wrapping action will be broken up into smaller intervals until the wrapping action is valid. For example, if you try to wrap a *paragraph* element with a *bold* element, it would not be valid, so the operation will wrap the text inside the paragraph instead, since it would be valid at that position.

• **RenameElementOperation**

This operation allows you to rename all occurrences of the elements identified by an XPath expression. The operation requires two parameters:

- **elementName** - The new element name
- **elementLocation** - The XPath expression that identifies the element occurrences to be renamed. If this parameter is missing, the operation renames the element at current caret position.

• **ExecuteTransformationScenariosOperation**

This operation allows running one or more transformation scenarios defined in the current document type association. It is useful to add to the toolbar buttons that trigger publishing to various output formats. The argument of the operation is:

- **scenarioNames** - The list of scenario names that will be executed, separated by new lines.

• **XSLTOperation** and **XQueryOperation**

Applies an XSLT or XQuery script on a source element and then replaces or inserts the result in a specified target element.

This operation has the following parameters:

• **sourceLocation**

An XPath expression indicating the element that the script will be applied on. If it is not defined then the element at the caret position will be used.

There may be situations in which you want to look at an ancestor of the current element and take decisions in the script based on this. In order to do this you can set the **sourceLocation** to point to an ancestor node (for example `/`) then declare a parameter called **currentElementLocation** in your script and use it to re-position in the current element like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
  xpath-default-namespace="http://docbook.org/ns/docbook"
  xmlns:saxon="http://saxon.sf.net/" exclude-result-prefixes="saxon">
  <!-- This is an XPath location which will be sent by the operation to the script -->
  <xsl:param name="currentElementLocation"/>

  <xsl:template match="/">
    <!-- Evaluate the XPath of the current element location -->
    <xsl:apply-templates
      select="saxon:eval(saxon:expression($currentElementLocation))"/>
  </xsl:template>

  <xsl:template match="para">
    <!-- And the context is again inside the current element,
        but we can use information from the entire XML -->
    <xsl:variable
      name="keyImage" select="//imagedata[@fileref='images/lake.jpeg']"
      /ancestor::inlinemediaobject/@xml:id/string()"/>
      <xref linkend="{$keyImage}" role="key_include"
        xmlns="http://docbook.org/ns/docbook">
        <xsl:value-of
          select="$currentElementLocation"></xsl:value-of>
      </xref>
    </xsl:template>
  </xsl:stylesheet>
```

• **targetLocation**

An XPath expression indicating the insert location for the result of the transformation. If it is not defined then the insert location will be at the caret.

- **script**

The script content (XSLT or XQuery). The base system ID for this will be the framework file, so any include/import reference will be resolved relative to the .framework file that contains this action definition.

For example, for the following script, the imported `xslt_operation.xsl` needs to be located in the current framework's directory.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:import href="xslt_operation.xsl"/>
</xsl:stylesheet>
```

- **action**

The insert action relative to the node determined by the target XPath expression. It can be: Replace, At caret position, Before, After, Inside as first child or Inside as last child.

- **caretPosition**

The position of the caret after the action is executed. It can be: Preserve, Before, Start, First editable position, End or After. If not specified the caret position can be specified by outputting in the XSLT script a `$(caret)` editor variable.

- **expandEditorVariables**

Parameter controlling the expansion of editor variables returned by the script processing. Expansion is enabled by default.

- **JSOperation**

Allows you to call the Java API from custom JavaScript content.

This operation has the following parameters:

- **script**

The JavaScript content to execute. It must have a function called `doOperation()`, which can use the predefined `authorAccess` variable. The `authorAccess` variable has access to the entire `ro.sync.ecss.extensions.api.AuthorAccess` Java API.

The following example is a script that can be used to move the caret location after the current element:

```
function doOperation(){
    caretOffset = authorAccess.getEditorAccess().getCaretOffset();
    currentNode = authorAccess.getDocumentController().getNodeAtOffset(caretOffset);
    //Move caret after current node
    authorAccess.getEditorAccess().setCaretPosition(currentNode.getEndOffset() + 1);
}
```

 **Note:** If you have a script called `commons.js` in the framework directory, you can call functions defined inside it from your custom script content so that you can use that external script file as a library of functions.

- **ExecuteMultipleActionsOperation**

This operation allows the execution of a sequence of actions, defined as a list of action IDs. The actions must be defined by the corresponding framework, or one of the common actions for all frameworks supplied by Oxygen XML Editor.

- **actionIDs** - The action IDs list which will be executed in sequence, the list must be a string sequence containing the IDs separated by new lines.

- **MoveElementOperation**

Flexible operation for moving an XML element to another location from the same document. XPath expressions are used to identify the source element and the target location. The operation takes the following parameters:

- **sourceLocation** - XPath expression that identifies the content to be moved.
- **deleteLocation** - XPath expression that identifies the node to be removed. This parameter is optional. If missing, the **sourceLocation** parameter will also identify the node to be deleted.
- **surroundFragment** - A string representation of an XML fragment. The moved node will be wrapped in this string before moving it in the destination.
- **targetLocation** - XPath expression that identifies the location where the node must be moved to.
- **insertPosition** - Argument that indicates the insert position.
- **moveOnlySourceContentNodes** - When `true`, only the content of the source element is moved.

• **ChangePseudoClassesOperation**

Operation that sets a list of pseudo class values to nodes identified by an XPath expression. It can also remove a list of values from nodes identified by an XPath expression. The operation accepts the following parameters:

- **setLocations** - An XPath expression indicating a list of nodes on which the specified list of pseudo classes will be set. If it is not defined, then the element at the caret position will be used.
- **setPseudoClassNames** - A space-separated list of pseudo class names which will be set on the matched nodes.
- **removeLocations** - An XPath expression indicating a list of nodes from which the specified list of pseudo classes will be removed. If it is not defined, then the element at the caret position will be used.
- **removePseudoClassNames** - A space-separated list of pseudo class names which will be removed from the matched nodes.

• **SetPseudoClassOperation**

An operation that sets a pseudo-class to an element. The operation accepts the following parameters:

- **elementLocation** - An XPath expression indicating the element on which the pseudo-class will be set. If it is not defined, then the element at caret position will be used.
- **name** - The pseudo-class local name.

Author operations can include parameters that contain the following editor variables:

- `$(caretp)` - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `$(selection)` - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `$(ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value'))` - To prompt for values at runtime, use the `ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')` editor variable. You can set the following parameters:
 - 'message' - The displayed message. Note the quotes that enclose the message.
 - type - Optional parameter, with one of the following values:

Parameter	
url	Format: <code>\$(ask('message', url, 'default_value'))</code> Description: Input is considered a URL. Oxygen XML Editor checks that the provided URL is valid. Example: <ul style="list-style-type: none"> • <code>\$(ask('Input URL', url))</code> - The displayed dialog box has the name Input URL. The expected input type is URL. • <code>\$(ask('Input URL', url, 'http://www.example.com'))</code> - The displayed dialog box has the name Input URL. The expected input type is URL. The input field displays the default value <code>http://www.example.com</code>.

Parameter	
password	<p>Format: \${ask('message', password, 'default')}</p> <p>Description: The input is hidden with bullet characters.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Input password', password)} - The displayed dialog box has the name 'Input password' and the input is hidden with bullet symbols. • \${ask('Input password', password, 'abcd')} - The displayed dialog box has the name 'Input password' and the input hidden with bullet symbols. The input field already contains the default abcd value.
generic	<p>Format: \${ask('message', generic, 'default')}</p> <p>Description: The input is considered to be generic text that requires no special handling.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Hello world!')} - The dialog box has a Hello world! message displayed. • \${ask('Hello world!', generic, 'Hello again!')} - The dialog box has a Hello world! message displayed and the value displayed in the input box is 'Hello again!'.
relative_url	<p>Format: \${ask('message', relative_url, 'default')}</p> <p>Description: Input is considered a URL. Oxygen XML Editor tries to make the URL relative to that of the document you are editing.</p> <p> Note: If the \$ask editor variable is expanded in content that is not yet saved (such as an <i>untitled</i> file, whose path cannot be determined), then Oxygen XML Editor will transform it into an absolute URL.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('File location', relative_url, 'C:/example.txt')} - The dialog box has the name 'File location'. The URL inserted in the input box is made relative to the current edited document location.
combobox	<p>Format: \${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</p> <p>Description: Displays a dialog box that offers a drop-down list. The drop-down list is populated with the given rendered_value values. Choosing such a value will return its associated value (real_value).</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems. The associated value will be returned based upon your selection. <p> Note: In this example Mac OS X is the default selected value and if selected it would return osx for the output.</p>

Parameter	
editable_combobox	<p>Format: \${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</p> <p>Description: Displays a dialog box that offers a drop-down list with editable elements. The drop-down list is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated real value (<code>real_value</code>) or the value inserted when you edit a list entry.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
radio	<p>Format: \${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</p> <p>Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a '<code>rendered_value</code>' and will return an associated '<code>real_value</code>'.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name 'Operating System'. The radio button group allows you to choose between the three operating systems. <p> Note: In this example Mac OS X is the default selected value and if selected it would return osx for the output.</p>

- '`default-value`' - optional parameter. Provides a default value.
- `${timeStamp}` - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- `${uuid}` - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java `UUID` class.
- `${id}` - Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- `${cfn}` - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- `${cfne}` - Current file name with extension. The current file is the one currently opened and selected.
- `${cf}` - Current file as file path, that is the absolute file path of the current edited document.
- `${cfld}` - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- `${frameworksDir}` - The path (as file path) of the [OXYGEN_DIR] / frameworks directory.
- `${pd}` - Current project folder as file path. Usually the current folder selected in the Project View.
- `${oxygenInstallDir}` - Oxygen XML Editor installation folder as file path.
- `${homeDir}` - The path (as file path) of the user home folder.
- `${pn}` - Current project name.

- `${env(VAR_NAME)}` - Value of the `VAR_NAME` environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the `${system(var.name)}` editor variable.
 - `${system(var.name)}` - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the `${env(VAR_NAME)}` editor variable instead.
 - `${date(pattern)}` - Current date. The allowed patterns are equivalent to the ones in the [Java SimpleDateFormat class](#).
- Example:** `YYYY-MM-dd;`



Note: This editor variable supports both the `xs:date` and `xs:datetime` parameters. For details about `xs:date`, go to <http://www.w3.org/TR/xmlschema-2/#date>. For details about `xs:datetime`, go to <http://www.w3.org/TR/xmlschema-2/#dateTime>.

The arguments of `InsertFragmentOperation` operation

fragment

This argument has a textual value. This value is parsed by Oxygen XML Editor as it was already in the document at the caret position. You can use entity references declared in the document and it is namespace aware. The fragment may have multiple roots.

You can even use namespace prefixes that are not declared in the inserted fragment, if they are declared in the document where the insertion is done. For the sake of clarity, you should always prefix and declare namespaces in the inserted fragment!

If the fragment contains namespace declarations that are identical to those found in the document, the namespace declaration attributes will be removed from elements contained by the inserted fragment.

There are two possible scenarios:

1. Prefixes that are not bound explicitly

For instance, the fragment:

```
<x:item id="dty2"/>
&ent;
<x:item id="dty3"/>
```

Can be correctly inserted in the document: ('|' marks the insertion point):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE x:root [
    <!ENTITY ent "entity">
]>

<x:root xmlns:x="nsp">
|
</x:root>
```

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE x:root [
    <!ENTITY ent "entity">
]>
<x:root xmlns:x="nsp">
    <x:item id="dty2"/>
    &ent;
    <x:item id="dty3"/>
</x:root>
```

2. Default namespaces

If there is a default namespace declared in the document and the document fragment does not declare a namespace, the elements from the fragment are considered to be in **no namespace**.

For instance the fragment:

```
<item id="dty2"/>
<item id="dty3"/>
```

Inserted in the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
|
</root>
```

Gives the result document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
    <item xmlns="" id="dty2"/>
    <item xmlns="" id="dty3"/>
</root>
```

insertLocation

An XPath expression that is relative to the current node. It selects the reference node for the fragment insertion.

insertPosition

One of the three constants: "**Inside**", "**After**", or "**Before**" , showing where the insertion is made relative to the reference node selected by the `insertLocation`. "**Inside**" has the meaning of the first child of the reference node.

goToNextEditablePosition

After inserting the fragment, the first editable position is detected and the caret is placed at that location. It handles any in-place editors used to edit attributes. It will be ignored if the fragment specifies a caret position using the `caret` editor variable. The possible values of this action are **true** and **false**.

The arguments of `SurroundWithFragmentOperation`

The Author operation `SurroundWithFragmentOperation` has only one argument:

- `fragment` -

The XML fragment that will surround the selection. For example let's consider the fragment:

```
<F>
  <A></A>
  <B>
    <C></C>
  </B>
</F>
```

and the document:

```
<doc>
  <X></X>
  <Y></Y>
  <Z></Z>
<doc>
```

Considering the selected content to be surrounded is the sequence of elements X and Y, then the result is:

```
<doc>
  <F>
    <A>
      <X></X>
      <Y></Y>
    </A>
    <B>
      <C></C>
    </B>
  </F>
  <Z></Z>
<doc>
```

Because the element A was the first leaf in the fragment, it received the selected content. The fragment was then inserted in the place of the selection.

Add a Custom Operation to an Existing Framework

This task explains how to add a custom Author operation to an existing document type.

1. Setup an Author sample project following [this set of instructions](#). The framework project is **oxygen-sample-framework**.
2. A number of classes in the *simple.documentation.framework.operations* package implement the *ro.sync.ecss.extensions.api.AuthorOperation* interface. Depending on your use-case, modify one of these classes.
3. Pack the operation class inside a Java jar library.
4. Copy the jar library to the [OXYGEN_DIR] / frameworks / [FRAMEWORK_DIR] directory.
5. [Open the Preferences dialog box](#), go to **Document Type Association**, and edit the document type (you need write access to the [OXYGEN_DIR]).
 - a) In the **Classpath** tab, add a new entry like: \${framework}/customAction.jar.
 - b) In the **Author** tab, add a new action which uses your custom operation.
 - c) Mount the action to the toolbars or menus.
6. Share the modifications with your colleagues. The files which should be shared are your *customAction.jar* library and the .framework configuration file from the [OXYGEN_DIR] / frameworks / [FRAMEWORK_DIR] directory.

Using Retina/HiDPI Images in Author Mode

Oxygen XML Editor provides support for Retina and HiDPI images through simple naming conventions. The higher resolution images are stored in the same images folder as the normal resolution images and they are identified by a scaling factor that is included in the name of the image files. For instance, images with a Retina scaling factor of 2 will include @2x in the name (for example, *myImage@2x.png*).

You can reference an image to style an element in a CSS by using the *url* function in the *content* property, as in the following example:

```
listItem:before{
    content: url('../img/myImage.png');
}
```

This would place the image that is loaded from the *myImage.png* file just before the *listItem* element. However, if you are using a Retina display (on a Mac), the icon looks a bit blurry as it automatically gets scaled, or if you are using an HiDPI display (on a Windows-based PC), the icon remains at the original size, thus it will look very small. To solve this rendering problem you need to be able to reference both a *normal* DPI image and a *high* DPI image. However, referencing both of them from the CSS is not practical, as there is no standard way of doing this.

Starting with version 17, Oxygen XML Editor interprets the argument of the *url* function as key rather than a fixed URL. Therefore, when running on a system with a Retina or HiDPI display, Oxygen XML Editor will first try to find the image file that corresponds to the retina scaling factor. For instance, using the previous example, Oxygen XML Editor would first try to find *myImage@2x.png*. If this file is not found, it defaults back to the *normal* resolution image file (*myImage.png*).

Oxygen XML Editor also supports dark color themes. This means that the background of the editor area can be of a dark color and the foreground a lighter color. On a dark background, you may find it useful to invert the colors of images. Again, this can be done with simple naming conventions. If an image designed for a dark background is not found, the *normal* image is used.

Retina/HiDPI Naming Convention

Refer to the following table for examples of the Retina/HiDPI image naming convention that is used in Oxygen XML Editor:

Color Theme	Referred Image File	Double Density Image File	Triple Density Image File
normal	./img/myImage.png	./img/myImage@2x.png	./img/myImage@3x.png

dark

./img/myImage_dark.png ./img/myImage_dark@2x.png ./img/myImage_dark@3x.png

Adding Retina/HiDPI Icons in a Framework

Higher resolution icons can also be included in customized frameworks for rendering them in a Retina or HiDPI display. The icons can be referenced directly from the Document Type customization (from the [Action dialog box](#)) or from an API (`ro.sync.ecml.workspace.api.node.customizer.XMLNodeRendererCustomizer`).

As with any image, the higher resolution icons are stored in the same images folder as the normal resolution images and they are identified by a scaling factor that is included in the name of the image files. For instance, icons with a Retina scaling factor of 2 will include `@2x` in the name (for example, `myIcon@2x.png`).

Developers should not specify the path of the alternate icons (`@2x` or `@3x`) in the [Action dialog box](#) or the [XMLNodeRendererCustomizer API](#). When using a Retina or HiDPI display, Oxygen XML Editor automatically searches the folder of the *normal* icon for a corresponding image file with a Retina scaling factor in the name. If the higher resolution icon file does not exist, the *normal* icon is scaled and used instead.

Java API - Extending Author Functionality through Java

Oxygen XML Editor Author has a built-in set of operations covering the insertion of text and XML fragments (see the [Author Default Operations](#)) and the execution of XPath expressions on the current document edited in Author mode. However, there are situations in which you need to extend this set. For instance if you need to enter an element whose attributes should be edited by the user through a graphical user interface. Or the users must send the selected element content or even the whole document to a server, for some kind of processing or the content authors must extract pieces of information from a server and insert it directly into the edited XML document. Or you need to apply an XPath expression on the current Author document and process the nodes of the result node set.

The following sections contain the Java programming interface (API) available to the developers. You will need the [Oxygen SDK](#) available [on the Oxygen XML Editor website](#) which includes the source code of the Author operations in the predefined document types and the full documentation in Javadoc format of the public API available for the developer of Author custom actions.

The next Java examples are making use of AWT classes. If you are developing extensions for the Oxygen XML Editor XML Editor plugin for Eclipse you will have to use their SWT counterparts.

It is assumed you already read the [Configuring Actions, Menus, Toolbar](#) section and you are familiar with the Oxygen XML Editor Author customization. You can find the XML schema, CSS and XML sample in the [Example Files Listings](#).



Attention:

Make sure the Java classes of your custom Author operations are compiled with the same Java version used by Oxygen XML Editor. Otherwise the classes may not be loaded by the Java virtual machine. For example if you run Oxygen XML Editor with a Java 1.6 virtual machine but the Java classes of your custom Author operations are compiled with a Java 1.7 virtual machine then the custom operations cannot be loaded and used by the Java 1.6 virtual machine.

Example 1. Simple Use of a Dialog Box from an Author Operation.

Let's start adding functionality for inserting images in the **Simple Documentation Framework** (shortly SDF). The images are represented by the `image` element. The location of the image file is represented by the value of the `href` attribute. In the Java implementation you will show a dialog box with a text field, in which the user can enter a full URL, or he can browse for a local file.

1. Setup an Author sample project following [this set of instructions](#). The framework project is `oxygen-sample-framework`.
2. Modify the `simple.documentation.framework.InsertImageOperation` class that implements the `ro.sync.ecss.extensions.api.AuthorOperation` interface. This interface defines three methods: `doOperation`, `getArguments` and `getDescription`

A short description of these methods follows:

- The `doOperation` method is invoked when the action is performed either by pressing the toolbar button, by selecting the menu item or by pressing the shortcut key. The arguments taken by this methods can be one of the following combinations:
 - an object of type `ro.sync.ecss.extensions.api.AuthorAccess` and a map
 - argument names and values
- The `getArguments` method is used by Oxygen XML Editor when the action is configured. It returns the list of arguments (name and type) that are accepted by the operation.
- The `getDescription` method is used by Oxygen XML Editor when the operation is configured. It returns a description of the operation.

Here is the implementation of these three methods:

```
/*
 * Performs the operation.
 */
public void doOperation(
    AuthorAccess authorAccess,
    ArgumentsMap arguments)
throws IllegalArgumentException,
    AuthorOperationException {
}

JFrame oxygenFrame = (JFrame) authorAccess.getWorkspaceAccess().getParentFrame();
String href = displayURLDialog(oxygenFrame);
if (href.length() != 0) {
    // Creates the image XML fragment.
    String imageFragment =
        "<image xmlns='http://www.oxygenxml.com/sample/documentation' href='"
        + href + "'/>";

    // Inserts this fragment at the caret position.
    int caretPosition = authorAccess.getEditorAccess().getCaretoffset();
    authorAccess.getDocumentController().insertXMLFragment(imageFragment, caretPosition);
}

/**
 * Has no arguments.
 *
 * @return null.
 */
public ArgumentDescriptor[] getArguments() {
    return null;
}

/**
 * @return A description of the operation.
 */
public String getDescription() {
    return "Inserts an image element. Asks the user for a URL reference.";
}
```

 **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK*, available as a Maven archetype [on the Oxygen XML Editor website](#).

 **Important:**

Make sure you always specify the namespace of the inserted fragments.

```
<image xmlns='http://www.oxygenxml.com/sample/documentation'
    href='path/to/image.png'/>
```

- Package the compiled class into a jar file. An example of an ANT script that packages the `classes` folder content into a jar archive named `sdf.jar` is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project" default="dist">
    <target name="dist">
        <jar destfile="sdf.jar" basedir="classes">
            <fileset dir="classes">
                <include name="**/*"/>
            </fileset>
        </jar>
    </target>
</project>
```

```
</target>
</project>
```

4. Copy the sdf.jar file into the frameworks/sdf folder.
5. Add the sdf.jar to the Author class path. To do this, [open the Preferences dialog box](#), go to **Document Type Association**, select SDF, and press the **Edit** button.
6. Select the **Classpath** tab in the lower part of the dialog box and press the **+ Add** button. In the displayed dialog box, enter the location of the jar file, relative to the Oxygen XML Editor frameworks folder.
7. Let's create now the action which will use the defined operation. Click on the **Actions** label. Copy the icon files for the menu item and for the toolbar in the frameworks/sdf folder.
8. Define the action's properties:
 - Set **ID** to **insert_image**.
 - Set **Name** to **Insert image**.
 - Set **Menu access key** to letter **i**.
 - Set **Toolbar action** to \${framework}/toolbarImage.png.
 - Set **Menu icon** to \${framework}/menuImage.png.
 - Set **Shortcut key** to **Ctrl (Meta on Mac OS)+Shift+i**.
9. Now let's set up the operation. You want to add images only if the current element is a section, book or article.
 - Set the value of **XPath expression** to


```
local-name()='section' or local-name()='book'
          or local-name()='article'
```
 - Set the **Invoke operation** field to simple.documentation.framework.InsertImageOperation.

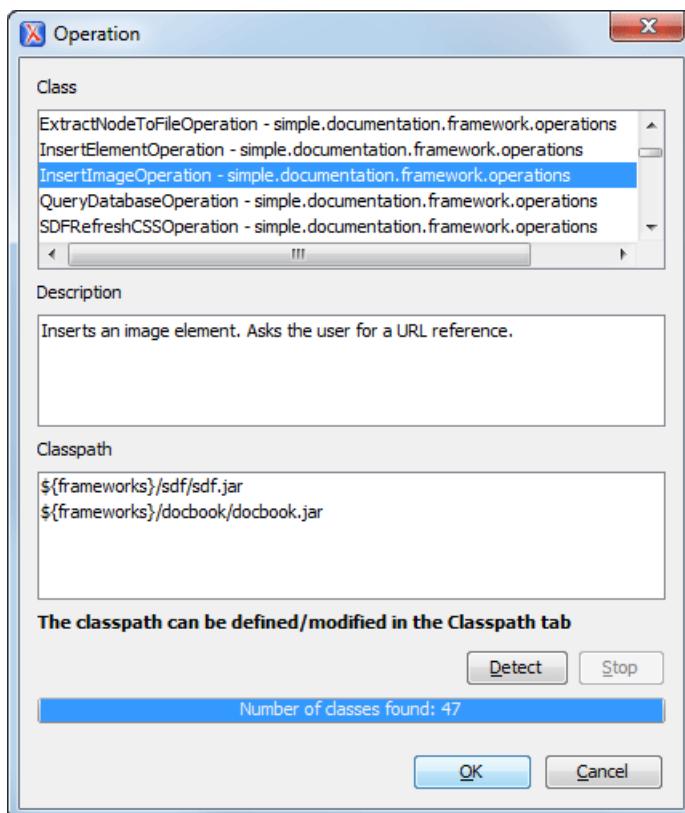


Figure 256: Selecting the Operation

10. Add the action to the toolbar, using the **Toolbar** panel.

To test the action, you can open the `sdf_sample.xml` sample, then place the caret inside a section between two `para` elements for instance. Press the button associated with the action from the toolbar. In the dialog box, select an image URL and press **OK**. The image is inserted into the document.

Example 2. Operations with Arguments. Report from Database Operation.

In this example you will create an operation that connects to a relational database and executes an SQL statement. The result should be inserted in the edited XML document as a `table`. To make the operation fully configurable, it will have arguments for the *database connection string*, the *user name*, the *password* and the *SQL expression*.

1. Setup an Author sample project following [this set of instructions](#). The framework project is `oxygen-sample-framework`.
2. Create the class `simple.documentation.framework.QueryDatabaseOperation`. This class must implements the `ro.sync.ecss.extensions.api.AuthorOperation` interface.

```
import ro.sync.ecss.extensions.api.ArgumentDescriptor;
import ro.sync.ecss.extensions.api.ArgumentsMap;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperation;
import ro.sync.ecss.extensions.api.AuthorOperationException;

public class QueryDatabaseOperation implements AuthorOperation{
```

3. Now define the operation's arguments. For each of them you will use a `String` constant representing the argument name:

```
private static final String ARG_JDBC_DRIVER = "jdbc_driver";
private static final String ARG_USER = "user";
private static final String ARG_PASSWORD = "password";
private static final String ARG_SQL = "sql";
private static final String ARG_CONNECTION = "connection";
```

4. You must describe each of the argument name and type. To do this implement the `getArguments` method which will return an array of argument descriptors:

```
public ArgumentDescriptor[] getArguments() {
    ArgumentDescriptor args[] = new ArgumentDescriptor[] {
        new ArgumentDescriptor(
            ARG_JDBC_DRIVER,
            ArgumentDescriptor.TYPE_STRING,
            "The name of the Java class that is the JDBC driver."),
        new ArgumentDescriptor(
            ARG_CONNECTION,
            ArgumentDescriptor.TYPE_STRING,
            "The database URL connection string."),
        new ArgumentDescriptor(
            ARG_USER,
            ArgumentDescriptor.TYPE_STRING,
            "The name of the database user."),
        new ArgumentDescriptor(
            ARG_PASSWORD,
            ArgumentDescriptor.TYPE_STRING,
            "The database password."),
        new ArgumentDescriptor(
            ARG_SQL,
            ArgumentDescriptor.TYPE_STRING,
            "The SQL statement to be executed.")
    };
    return args;
}
```

These names, types and descriptions will be listed in the **Arguments** table when the operation is configured.

5. When the operation is invoked, the implementation of the `doOperation` method extracts the arguments, forwards them to the method that connects to the database and generates the XML fragment. The XML fragment is then inserted at the caret position.

```
public void doOperation(AuthorAccess authorAccess, ArgumentsMap map)
    throws IllegalArgumentException, AuthorOperationException {

    // Collects the arguments.
    String jdbcDriver =
        (String)map.getArgumentValue(ARG_JDBC_DRIVER);
    String connection =
```

```

(String)map.getArgumentValue(ARG_CONNECTION);
String user =
(String)map.getArgumentValue(ARG_USER);
String password =
(String)map.getArgumentValue(ARG_PASSWORD);
String sql =
(String)map.getArgumentValue(ARG_SQL);

int caretPosition = authorAccess.getCaretOffset();
try {
authorAccess.getDocumentController().insertXMLFragment(
    getFragment(jdbcDriver, connection, user, password, sql),
    caretPosition);
} catch (SQLException e) {
throw new AuthorOperationException(
    "The operation failed due to the following database error: "
    + e.getMessage(), e);
} catch (ClassNotFoundException e) {
throw new AuthorOperationException(
    "The JDBC database driver was not found. Tried to load '"
    + jdbcDriver + "'", e);
}
}
}

```

6. The `getFragment` method loads the JDBC driver, connects to the database and extracts the data. The result is a `table` element from the `http://www.oxygenxml.com/sample/documentation` namespace. The `header` element contains the names of the SQL columns. All the text from the XML fragment is escaped. This means that the '<' and '&' characters are replaced with the '<' and '&' character entities to ensure the fragment is well-formed.

```

private String getFragment(
    String jdbcDriver,
    String connectionURL,
    String user,
    String password,
    String sql) throws
    SQLException,
    ClassNotFoundException {

    Properties pr = new Properties();
    pr.put("characterEncoding", "UTF8");
    pr.put("useUnicode", "TRUE");
    pr.put("user", user);
    pr.put("password", password);

    // Loads the database driver.
    Class.forName(jdbcDriver);
    // Opens the connection
    Connection connection =
        DriverManager.getConnection(connectionURL, pr);
    java.sql.Statement statement =
        connection.createStatement();
    ResultSet resultSet =
        statement.executeQuery(sql);

    StringBuffer fragmentBuffer = new StringBuffer();
    fragmentBuffer.append(
        "<table xmlns=" +
        "'http://www.oxygenxml.com/sample/documentation'");

    /**
     * Creates the table header.
     */
    fragmentBuffer.append("<header>");
    ResultSetMetaData metaData = resultSet.getMetaData();
    int columnCount = metaData.getColumnCount();
    for (int i = 1; i <= columnCount; i++) {
        fragmentBuffer.append("<td>");
        fragmentBuffer.append(
            xmlEscape(metaData.getColumnName(i)));
        fragmentBuffer.append("</td>");
    }
    fragmentBuffer.append("</header>");

    /**
     * Creates the table content.
     */
    while (resultSet.next()) {
        fragmentBuffer.append("<tr>");
        for (int i = 1; i <= columnCount; i++) {
            fragmentBuffer.append("<td>");
            fragmentBuffer.append(
                xmlEscape(resultSet.getObject(i)));
            fragmentBuffer.append("</td>");
        }
    }
}
}

```

```

        fragmentBuffer.append("</tr>");
    }

    fragmentBuffer.append("</table>");

    // Cleanup
    resultSet.close();
    statement.close();
    connection.close();
    return fragmentBuffer.toString();
}

```



Note: The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK*, available as a Maven archetype [on the Oxygen XML Editor website](#).

7. Package the compiled class into a jar file.
8. Copy the jar file and the JDBC driver files into the frameworks/sdf directory.
9. Add the jars to the Author class path. To do this, open the **Document Type** dialog box, select **SDF** and press the **Edit** button. Select the **Classpath** tab in the lower part of the dialog box.
10. Click on the **Actions** label. The action properties are:
 - Set **ID** to **clients_report**.
 - Set **Name** to **Clients Report**.
 - Set **Menu access key** to letter r.
 - Set **Description** to **Connects to the database and collects the list of clients**.
 - Set **Toolbar icon** to \${framework}/TableDB20.png (image TableDB20.png is already stored in the frameworks / sdf folder).
 - Leave empty the **Menu icon**.
 - Set **shortcut key** to **Ctrl Shift C (Command Shift C on OS X)**.
11. The action will work only if the current element is a **section**. Set up the operation as follows:
 - Set **XPath expression** to:


```
local-name()='section'
```
 - Use the Java operation defined earlier to set the **Invoke operation** field. Press the **Choose** button, then select `simple.documentation.framework.QueryDatabaseOperation`. Once selected, the list of arguments is displayed. In the figure below the first argument, `jdbc_driver`, represents the class name of the MySQL JDBC driver. The connection string has the URL syntax : `jdbc://<database_host>:<database_port>/<database_name>`.

The SQL expression used in the example follows, but it can be any valid SELECT expression which can be applied to the database:

```
SELECT userID, email FROM users
```

12. Add the action to the toolbar, using the **Toolbar** panel.

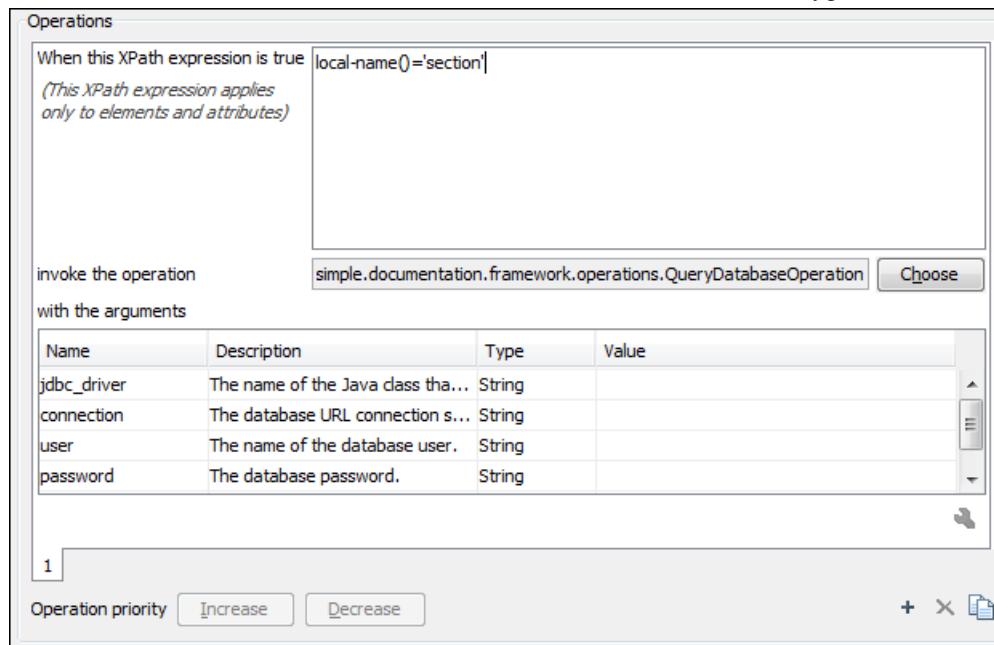


Figure 257: Java Operation Arguments Setup

To test the action you can open the `sdf_sample.xml` sample place the caret inside a `section` between two para elements for instance. Press the **Create Report** button from the toolbar. You can see below the toolbar with the action button and sample table inserted by the **Clients Report** action.

The screenshot shows a table extracted from a database. The table has two columns: 'userID' and 'Author'. The data is as follows:

userID	Author
204473	michael@test-oxy.ro
204477	mary@test-oxy.ro
204478	adrian@test-oxy.ro
204479	will@test-oxy.ro

Figure 258: Table Content Extracted from the Database

Editing Attributes In-Place Using Form Controls

To edit attributes in the **Author** mode, use the [Attributes View](#).

The `oxy_editor` CSS extension function allows you to edit attribute and element text values directly in the **Author** mode using form-based controls. Various implementations are available out-of-the-box: [combo boxes](#), [checkboxes](#), [text fields](#), [pop-ups](#), [buttons](#), which invoke custom **Author** actions or [URL choosers](#). You can also implement custom editors for your specific needs.

As a working example, the bundled *samples* project contains a file called `personal.xml`, which allows you to edit attributes in-place using some of these default implementations.

Localizing Frameworks

Oxygen XML Editor supports framework localization (translating framework actions, buttons, and menu entries to different languages). This lets you develop and distribute a framework to users that speak different languages without changing the distributed framework. Changing the language used in Oxygen XML Editor in the Global preferences page is enough to set the right language for each framework.

To localize the content of a framework, create a `translation.xml` file which contains all the translation (key, value) mappings. The `translation.xml` has the following format:

```
<translation>
  <languageList>
    <language description="English" lang="en_US"/>
    <language description="German" lang="de_DE"/>
    <language description="French" lang="fr_FR"/>
  </languageList>
  <key value="list">
    <comment>List menu item name.</comment>
    <val lang="en_US">List</val>
    <val lang="de_DE">Liste</val>
    <val lang="fr_FR">Liste</val>
  </key>
  .....
</translation>
```

Oxygen XML Editor matches the GUI language with the language set in the `translation.xml` file. If this language is not found, the first available language declared in the `languageList` tag for the corresponding framework is used.

Add the directory where this file is located to the **Classpath** list corresponding to the edited document type.

After you create this file, you are able to use the keys defined in it to customize the name and description of the following:

- framework actions
- menu entries
- contextual menus
- toolbars
- static CSS content

For example, if you want to localize the bold action, [open the Preferences dialog box](#) and go to **Document Type Association**. Use the **New** or **Edit** button to open the **Document type** dialog box, go to **Author > Actions**, and rename the bold action to `${i18n(translation_key)}`. Actions with a name format different than `${i18n(translation_key)}` are not localized. `translation_key` corresponds to the key from the `translation.xml` file.

Now open the `translation.xml` file and edit the translation entry if it exists or create one if it does not exist. This example presents an entry in the `translation.xml` file:

```
<key value="translation_key">
  <comment>Bold action name.</comment>
  <val lang="en_US">Bold</val>
  <val lang="de_DE">Bold</val>
  <val lang="fr_FR">Bold</val>
</key>
```

To use a description from the `translation.xml` file in the Java code used by your custom framework, use the new `ro.sync.ecss.extensions.api.AuthorAccess.getAuthorResourceBundle()` API method to request the associated value for a certain key. This allows all the dialog boxes that you present from your custom operations to have labels translated in different languages.

You can also reference a key directly in the CSS content:

```
title:before{
  content:"${i18n(title.key)} : ";
```



Note: You can enter any language you want in the `languagelist` tag and any number of keys.

The `translation.xml` file for the DocBook framework is located here: `[OXYGEN_DIR]/frameworks/docbook/i18n/translation.xml`. In the **Classpath** list corresponding to the DocBook document type the following entry was added:

```
 ${framework}/i18n/.
```

To see how the DocBook actions are defined to use these keys for their name and description, [open the Preferences dialog box](#) and go to **Document Type Association > Author > Actions**. If you look in the Java class

`ro.sync.ecss.extensions.docbook.table.SADocbookTableCustomizerDialog` available in the `oxygen-sample-framework` module of the [Oxygen SDK](#) Maven archetype, you can see how the new `ro.sync.ecss.extensions.api.AuthorResourceBundle` API is used to retrieve localized descriptions for different keys.

How to Pack and Deploy an Add-on

Packing a Plugin or Framework as an Add-on

This procedure is suitable for developers who want a better control over the add-on package or those who want to automate some of the steps:

1. Pack the plugin or framework as a **ZIP** file or a *Java Archive (JAR)*. Please note that you should pack the entire root directory not just its contents.
2. Digitally sign the package. Please note that you can perform this step only if you have created a *JAR* at the previous step. You will need a certificate signed by a trusted authority. To sign the jar you can either use the `jarsigner` command line tool inside Oracle's Java Development Kit. (`[JDK_DIR]/bin/jarsigner.exe`) or, if you are working with *Apache Ant*, you can use the `signjar` task (which is just a front for the `jarsigner` command line tool).



Note: The benefit of having a signed add-on is that the user can verify the integrity of the add-on issuer. If you don't have such a certificate you can generate one yourself using the `keytool` command line tool. Please note that this approach is mostly recommended for tests since anyone can create a self signed certificate.

3. Create a descriptor file. You can use a template that Oxygen XML Editor provides. To use this template, go to **File > New** and select the **Oxygen add-ons update site** template. Once deployed, this descriptor file is referenced as *update site*.

Alternatively, you can use the **Add-ons Packager** plugin by following this procedure:

1. Install the **Add-ons Packager** plugin from <http://www.oxygenxml.com/InstData/Addons/optional/updateSite.xml> as described in the [Installing Add-ons procedure](#).
2. Restart Oxygen XML Editor. If the add-on is correctly installed, the **Add-ons packager** toolbar action is available.
3. Invoke the **Add-ons packager** toolbar action and input the required information in the displayed dialog box.
4. Press **OK** to complete the packaging process.

Deploying an Add-on

To deploy an add-on, copy the **ZIP/JAR** file and the descriptor file to an HTTP server. The URL to this location serves as the *Update Site URL*.

Creating the Basic Association

Let us go through an example of creating a document type and editing an XML document of this type. We will call our document type **Simple Documentation Framework**.

First Step - XML Schema

Our documentation framework will be very simple. The documents will be either articles or books, both composed of sections. The sections may contain titles, paragraphs, figures, tables and other sections. To complete the picture, each section will include a def element from another namespace.

The first schema file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.oxygenxml.com/sample/documentation"
  xmlns:doc="http://www.oxygenxml.com/sample/documentation"
  xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
  elementFormDefault="qualified">

  <xs:import namespace=
    "http://www.oxygenxml.com/sample/documentation/abstracts"
    schemaLocation=
    "abs.xsd"/>
```

The namespace of the documents will be `http://www.oxygenxml.com/sample/documentation`. The namespace of the def element is `http://www.oxygenxml.com/sample/documentation/abstracts`.

Now let's define the structure of the sections. They all start with a title, then have the optional def element then either a sequence of other sections, or a mixture of paragraphs, images and tables.

```
<xs:element name="book" type="doc:sectionType"/>
<xs:element name="article" type="doc:sectionType"/>
<xs:element name="section" type="doc:sectionType"/>

<xs:complexType name="sectionType">
  <xs:sequence>
    <xs:element name="title" type="xs:string"/>
    <xs:element ref="abs:def" minOccurs="0"/>
    <xs:choice>
      <xs:sequence>
        <xs:element ref="doc:section" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="doc:para"/>
        <xs:element ref="doc:image"/>
        <xs:element ref="doc:table"/>
      </xs:choice>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

The paragraph contains text and other styling markup, such as bold (**b**) and italic (*i*) elements.

```
<xs:element name="para" type="doc:paragraphType"/>

<xs:complexType name="paragraphType" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="b"/>
    <xs:element name="i"/>
  </xs:choice>
</xs:complexType>
```

The image element has an attribute with a reference to the file containing image data.

```
<xs:element name="image">
  <xs:complexType>
    <xs:attribute name="href" type="xs:anyURI" use="required"/>
  </xs:complexType>
</xs:element>
```

The table contains a header row and then a sequence of rows (`tr` elements) each of them containing the cells. Each cell has the same content as the paragraphs.

```
<xs:element name="table">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="header">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="td" maxOccurs="unbounded"
```

```

        type="doc:paragraphType"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="tr" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="td" type="doc:tdType"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="tdType">
    <xs:complexContent>
        <xs:extension base="doc:paragraphType">
            <xs:attribute name="row_span" type="xs:integer"/>
            <xs:attribute name="column_span" type="xs:integer"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

The def element is defined as a text only element in the imported schema abs .xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
        "http://www.oxygenxml.com/sample/documentation/abstracts">
    <xs:element name="def" type="xs:string"/>
</xs:schema>

```

Now the XML data structure will be styled.

Schema Settings

In *the dialog box for editing the document type properties*, in the bottom section there are a series of tabs. The first one refers to the schema that is used for validation of the documents that match the defined **Association Rules**.

- ! **Important:** If the document refers a schema, using for instance a DOCTYPE declaration or a xsi:schemaLocation attribute, the schema from the document type association will not be used when validating.

Schema Type

Select from the combo box the value **XML Schema**.

Schema URI

Enter the value \${frameworks} / sdf / schema / sdf .xsd. We should use the \${frameworks} editor variable in the schema URI path instead of a full path in order to be valid for different Oxygen XML Editor installations.

- ! **Important:** The \${frameworks} variable is expanded at the validation time into the absolute location of the directory containing the frameworks.

Second Step - The CSS

If you read the *Simple Customization Tutorial* then you already have some basic notions about creating simple styles. The example document contains elements from different namespaces, so you will use CSS Level 3 extensions supported by the Author layout engine to associate specific properties with that element.

Defining the General Layout

Now the basic layout of the rendered documents is created.

Elements that are stacked one on top of the other are: `book`, `article`, `section`, `title`, `figure`, `table`, `image`. These elements are marked as having `block` style for display. Elements that are placed one after the other in a flowing sequence are: `b`, `i`. These will have `inline` display.

```
/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
}
```



Important:

Having `block` display children in an `inline` display parent, makes Oxygen XML Editor Author change the style of the parent to `block` display.

Styling the `section` Element

The title of any section must be bold and smaller than the title of the parent section. To create this effect a sequence of CSS rules must be created. The `*` operator matches any element, it can be used to match titles having progressive depths in the document.

```
title{
    font-size: 2.4em;
    font-weight:bold;
}
* * title{
    font-size: 2.0em;
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}
```

It's useful to have before the title a constant text, indicating that it refers to a section. This text can include also the current section number. The `:before` and `:after` pseudo elements will be used, plus the CSS counters.

First declare a counter named `sect` for each `book` or `article`. The counter is set to zero at the beginning of each such element:

```
book,
article{
    counter-reset:sect;
}
```

The `sect` counter is incremented with each `section`, that is a direct child of a `book` or an `article` element.

```
book > section,
article > section{
    counter-increment:sect;
}
```

The "static" text that will prefix the section title is composed of the constant "Section ", followed by the decimal value of the `sect` counter and a dot.

```
book > section > title:before,
article > section > title:before{
    content: "Section " counter(sect) ". ";
}
```

To make the documents easy to read, you add a margin to the sections. In this way the higher nesting level, the larger the left side indent. The margin is expressed relatively to the parent bounds:

```
section{
    margin-left:1em;
    margin-top:1em;
}
```

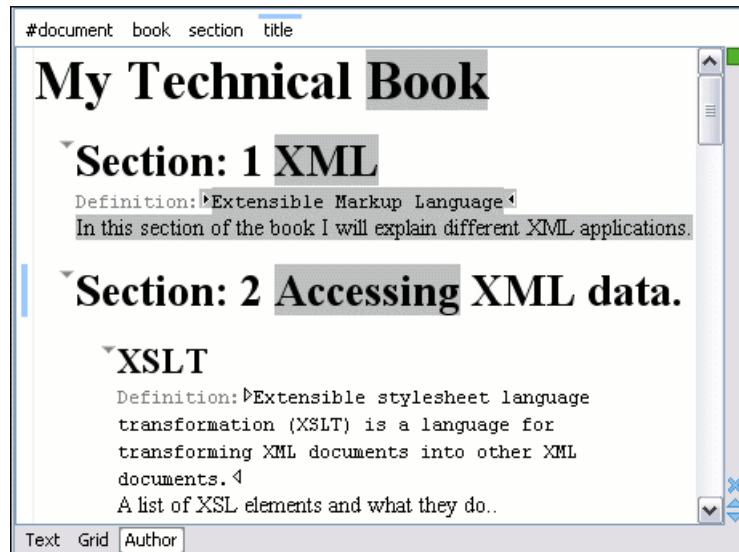


Figure 259: A sample of nested sections and their titles.

In the above screenshot you can see a sample XML document rendered by the CSS stylesheet. The selection "avoids" the text that is generated by the CSS "content" property. This happens because the CSS generated text is not present in the XML document and is just a visual aid.

Styling the Inline Elements

The "bold" style is obtained by using the `font-weight` CSS property with the value `bold`, while the "italic" style is specified by the `font-style` property:

```
b {
    font-weight:bold;
}

i {
    font-style:italic;
}
```

Styling Images

The CSS 2.1 does not specify how an element can be rendered as an image. To overpass this limitation, Oxygen XML Editor Author supports a CSS Level 3 extension allowing to load image data from an URL. The URL of the image must be specified by one of the element attributes and it is resolved through the catalogs specified in Oxygen XML Editor.

```
image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

Our `image` element has the required attribute `href` of type `xs:anyURI`. The `href` attribute contains an image location so the rendered content is obtained by using the function:

```
attr(href, url)
```

The first argument is the name of the attribute pointing to the image file. The second argument of the `attr` function specifies the type of the content. If the type has the `url` value, then Oxygen XML Editor identifies the content as being an image. If the type is missing, then the content will be the text representing the attribute value.

Oxygen XML Editor Author handles both absolute and relative specified URLs. If the image has an *absolute* URL location (for example: "http://www.oasis-open.org/images/standards/oasis_standard.jpg") then it is loaded directly from this location. If the image URL is *relative* specified to the XML document (for example: "images/my_screenshot.jpg") then the location is obtained by adding this value to the location of the edited XML document.

An image can also be referenced by the name of a DTD entity which specifies the location of the image file. For example if the document declares an entity **graphic** which points to a JPEG image file:

```
<!ENTITY graphic SYSTEM "depo/keyboard_shortcut.jpg" NDATA JPEG>
```

and the image is referenced in the XML document by specifying the name of the entity as the value of an attribute:

```
<mediaobject>
  <imageobject>
    <imagedata entityref="graphic" scale="50"/>
  </imageobject>
</mediaobject>
```

The CSS should use the functions `url`, `attr` and `unparsed-entity-uri` for displaying the image in the Author mode:

```
imagedata[entityref]{
  content: url(unparsed-entity-uri(attr(entityref)));
}
```

To take into account the value of the `width` attribute of the `imagedata` and use it for resizing the image, the CSS can define the following rule:

```
imagedata[width]{
  width:attr(width, length);
}
```

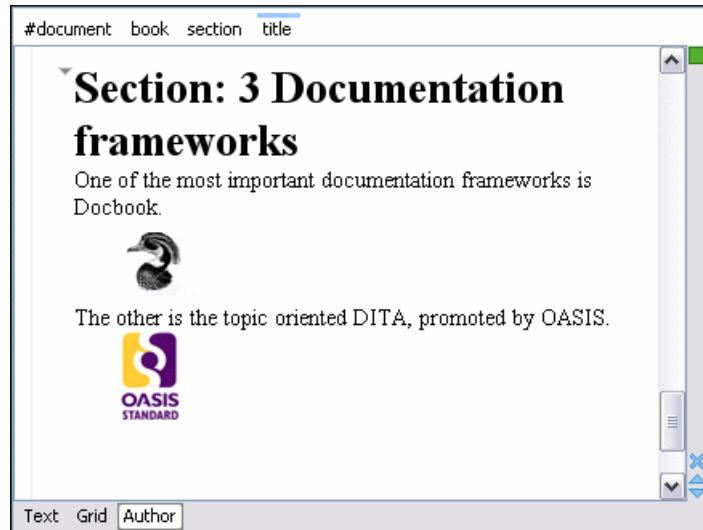


Figure 260: Samples of images in Author

Testing the Document Type Association

To test the new Document Type create an XML instance that is conforming with the *Simple Documentation Framework* association rules. You will not specify an XML Schema location directly in the document, using an

xsi:schemaLocation attribute; Oxygen XML Editor will detect instead its associated document type and use the specified schema.

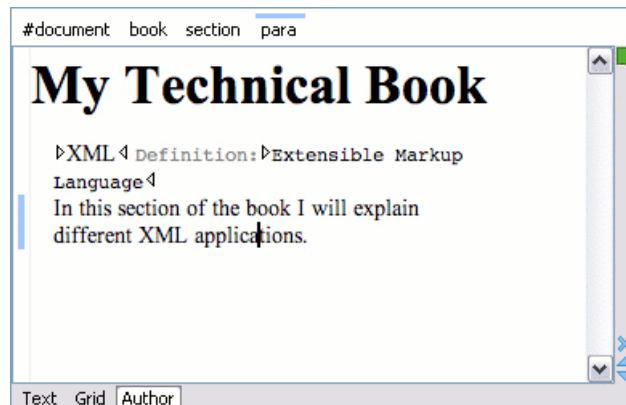
```
<book xmlns="http://www.oxygenxml.com/sample/documentation"
      xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">

    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will
              explain different XML applications.</para>
    </section>
</book>
```

When trying to validate the document there should be no errors. Now modify the title to title2. Validate again. This time there should be one error:

```
cvc-complex-type.2.4.a: Invalid content was found starting with element
'title2'. One of '{http://www.oxygenxml.com/sample/documentation}:title'
is expected.
```

Undo the tag name change. Press on the **Author** button at the bottom of the editing area. Oxygen XML Editor should load the CSS from the document type association and create a layout similar to this:



Organizing the Framework Files

First, create a new folder called sdf (from "Simple Documentation Framework") in [OXYGEN_DIR] / frameworks. This folder will be used to store all files related to the documentation framework. The following folder structure will be created:

```
oxygen
  frameworks
    sdf
      schema
      css
```

The frameworks directory is the container where all the Oxygen XML Editor framework customizations are located. Each subdirectory contains files related to a specific type of XML documents: schemas, catalogs, stylesheets, CSS stylesheets, etc. Distributing a framework means delivering a framework directory.

It is assumed that you have the right to create files and folder inside the Oxygen XML Editor installation directory. If you do not have this right, you will have to install another copy of the program in a folder you have access to, the home directory for instance, or your desktop. You can download the "all platforms" distribution from the oXygen website and extract it in the chosen folder.

To test your framework distribution, copy it in the frameworks directory of the newly installed application and start Oxygen XML Editor by running the provided start-up script files.

You should copy the created schema files abs . xsd and sdf . xsd, sdf . xsd being the master schema, to the schema directory and the CSS file sdf . css to the css directory.

Packaging and Deploying

Using a file explorer, go to the Oxygen XML Editor [OXYGEN_DIR] / frameworks directory. Select the sdf directory and make an archive from it. Move it to another Oxygen XML Editor installation (eventually on another computer). Extract it in the [OXYGEN_DIR] / frameworks directory. Start Oxygen XML Editor and test the association as explained above.

If you create multiple document type associations and you have a complex directory structure it might be easy from the deployment point of view to use an Oxygen XML Editor All Platforms distribution. Add your framework files to it, repackage it and send it to the content authors.

- ! **Attention:** When deploying your customized sdf directory please make sure that your sdf directory contains the sdf.framework file (that is the file defined as External Storage in the **Document Type Association** dialog box shall always be stored inside the sdf directory). If your external storage points somewhere else Oxygen XML Editor will not be able to update the Document Type Association options automatically on the deployed computers.

Configuring New File Templates

You will create a set of document templates that the content authors will use as starting points for creating *Simple Document Framework* books and articles.

Each Document Type Association can point to a directory, usually named templates, containing the file templates. All files found here are considered templates for the respective document type. The template name is taken from the file name, and the template type is detected from the file extension.

1. Go to the [OXYGEN_DIR] \ frameworks \ sdf directory and create a directory named templates. The directory tree of the documentation framework now is:

```

oxygen
  frameworks
    sdf
      schema
      css
      templates
  
```

2. In the templates directory create two files: a file for the *book* template and another one for the *article* template.

The Book.xml file:

```

<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
  <title>Book Template Title</title>
  <section>
    <title>Section Title</title>
    <abs:def/>
    <para>This content is copyrighted:</para>
    <table>
      <header>
        <td>Company</td>
        <td>Date</td>
      </header>
      <tr>
        <td></td>
        <td></td>
      </tr>
    </table>
  </section>
</book>
  
```

The Article.xml file:

```

<?xml version="1.0" encoding="UTF-8"?>
<article
  xmlns="http://www.oxygenxml.com/sample/documentation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <title></title>
  <section>
    <title></title>
    <para></para>
    <para></para>
  </section>
</article>
  
```

```
</section>
</article>
```

You can also use *editor variables* in the template files' content and they will be expanded when the files are opened.

 **Note:** You should avoid using the \${cf}, \${currentFileURL}, and \${cfdu} editor variables when you save your documents in a data base.

3. Open the **Document Type** dialog box for the **SDF** framework and click the **Templates** tab. In the **Templates directory** text field, introduce the \${frameworkDir}/templates path. As you have already seen before, it is recommended that all the file references made from a Document Type Association to be relative to the \${frameworkDir} directory. Binding a Document Type Association to an absolute file (e. g.: C:\some_dir\templates) makes the association difficult to share between users.
4. To test the templates settings, go to **File > New** to display the **New** document dialog box. The names of the two templates are prefixed with the name of the Document Type Association (**SDF** in this case). Selecting one of them should create a new XML file with the content specified in the template file.

Editor Variables

An editor variable is a shorthand notation for context-dependent information, such as a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example, the input URL of a transformation, the output file path of a transformation, or the command line of an external tool) to make a command or a parameter generic and re-usable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Editor commands of external engines or other external tools, in transformation scenarios, Author operations, and in validation scenarios:

- \${oxygenHome} - Oxygen XML Editor installation folder as URL.
- \${oxygenInstallDir} - Oxygen XML Editor installation folder as file path.
- \${framework} - The path (as URL) of the current framework, as part of the [OXYGEN_DIR]/frameworks directory.
- \${framework(fr_name)} - The path (as URL) of the fr_name framework.
- \${frameworkDir(fr_name)} - The path (as file path) of the fr_name framework.

 **Note:** Because multiple frameworks might have the same name (although it is not recommended), for both \${framework(fr_name)} and \${frameworkDir(fr_name)} editor variables Oxygen XML Editor employs the following algorithm when searching for a given framework name:

- all frameworks are sorted, from high to low, according to their **Priority** setting from the [Document Type Association preferences page](#). Only frameworks that have the **Enabled** checkbox set are taken into account.
- next, if the two or more frameworks have the same name and priority, a further sorting based on the **Storage** setting is made, in the exact following order:
 - frameworks stored in the internal Oxygen XML Editor options
 - additional frameworks added in the [Locations preferences page](#)
 - frameworks installed using the add-ons support
 - frameworks found in the [main frameworks location](#) (**Default** or **Custom**)
- \${frameworks} - The path (as URL) of the [OXYGEN_DIR] directory.
- \${frameworkDir} - The path (as file path) of the current framework, as part of the [OXYGEN_DIR]/frameworks directory.
- \${frameworksDir} - The path (as file path) of the [OXYGEN_DIR]/frameworks directory.
- \${home} - The path (as URL) of the user home folder.
- \${homeDir} - The path (as file path) of the user home folder.
- \${pdu} - Current project folder as URL. Usually the current folder selected in the Project View.

- `${pd}` - Current project folder as file path. Usually the current folder selected in the Project View.
- `${pn}` - Current project name.
- `${cfdu}` - Current file folder as URL, that is the path of the current edited document up to the name of the parent folder, represented as a URL.
- `${cfdu}` - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- `${cfn}` - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- `${cfne}` - Current file name with extension. The current file is the one currently opened and selected.
- `${cf}` - Current file as file path, that is the absolute file path of the current edited document.
- `${af}` - The local file path of the ZIP archive that includes the current edited document.
- `${afu}` - The URL path of the ZIP archive that includes the current edited document.
- `${afd}` - The local directory path of the ZIP archive that includes the current edited document.
- `${afdu}` - The URL path of the directory of the ZIP archive that includes the current edited document.
- `${afn}` - The file name (without parent directory and without file extension) of the zip archive that includes the current edited file.
- `${afne}` - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the current edited file.
- `${currentFileURL}` - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- `${ps}` - Path separator, that is the separator which can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- `${timeStamp}` - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- `${caret}` - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `${selection}` - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `${id}` - Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- `${uuid}` - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java `UUID` class.
- `${env(VAR_NAME)}` - Value of the `VAR_NAME` environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the `${system(var.name)}` editor variable.
- `${system(var.name)}` - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the `${env(VAR_NAME)}` editor variable instead.
- `${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}` - To prompt for values at runtime, use the `${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')}` editor variable. You can set the following parameters:
 - `'message'` - The displayed message. Note the quotes that enclose the message.
 - `type` - Optional parameter, with one of the following values:

Parameter	
<code>url</code>	<p>Format: <code> \${ask('message', url, 'default_value')} </code></p> <p>Description: Input is considered a URL. Oxygen XML Editor checks that the provided URL is valid.</p>
	<p>Example:</p> <ul style="list-style-type: none"> • <code> \${ask('Input URL', url)} </code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL.

Parameter	
	<ul style="list-style-type: none"> • \${ask('Input URL', url, 'http://www.example.com')} - The displayed dialog box has the name Input URL. The expected input type is URL. The input field displays the default value http://www.example.com.
password	<p>Format: \${ask('message', password, 'default')}</p> <p>Description: The input is hidden with bullet characters.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Input password', password)} - The displayed dialog box has the name 'Input password' and the input is hidden with bullet symbols. • \${ask('Input password', password, 'abcd')} - The displayed dialog box has the name 'Input password' and the input hidden with bullet symbols. The input field already contains the default abcd value.
generic	<p>Format: \${ask('message', generic, 'default')}</p> <p>Description: The input is considered to be generic text that requires no special handling.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Hello world!')} - The dialog box has a Hello world! message displayed. • \${ask('Hello world!', generic, 'Hello again!')} - The dialog box has a Hello world! message displayed and the value displayed in the input box is 'Hello again!'.
relative_url	<p>Format: \${ask('message', relative_url, 'default')}</p> <p>Description: Input is considered a URL. Oxygen XML Editor tries to make the URL relative to that of the document you are editing.</p> <p> Note: If the \$ask editor variable is expanded in content that is not yet saved (such as an <i>untitled</i> file, whose path cannot be determined), then Oxygen XML Editor will transform it into an absolute URL.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('File location', relative_url, 'C:/example.txt')} - The dialog box has the name 'File location'. The URL inserted in the input box is made relative to the current edited document location.
combobox	<p>Format: \${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</p> <p>Description: Displays a dialog box that offers a drop-down list. The drop-down list is populated with the given rendered_value values. Choosing such a value will return its associated value (real_value).</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems. The associated value will be returned based upon your selection.

Parameter	
	 Note: In this example Mac OS X is the default selected value and if selected it would return osx for the output.
editable_combobox	<p>Format: \${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</p> <p>Description: Displays a dialog box that offers a drop-down list with editable elements. The drop-down list is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated real value (<code>real_value</code>) or the value inserted when you edit a list entry.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
radio	<p>Format: \${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</p> <p>Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a '<code>rendered_value</code>' and will return an associated '<code>real_value</code>'.</p> <p>Example:</p> <ul style="list-style-type: none"> • \${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name 'Operating System'. The radio button group allows you to choose between the three operating systems. <p> Note: In this example Mac OS X is the default selected value and if selected it would return osx for the output.</p>

- '`default-value`' - optional parameter. Provides a default value.
- `$(date(pattern))` - Current date. The allowed patterns are equivalent to the ones in the [Java SimpleDateFormat class](#).
Example: `yyyy-MM-dd`;

 **Note:** This editor variable supports both the `xs:date` and `xs:datetime` parameters. For details about `xs:date`, go to <http://www.w3.org/TR/xmlschema-2/#date>. For details about `xs:datetime`, go to <http://www.w3.org/TR/xmlschema-2/#dateTime>.
- `$/dbgXML` - The local file path to the XML document which is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- `$/dbgXSL` - The local file path to the XSL/XQuery document which is current selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- `$/tsf` - The transformation result file path. If the current opened file has an associated scenario which specifies a transformation output file, this variable expands to it.
- `$/dsu` - The path of the detected schema as an URL for the current validated XML document.
- `$/ds` - The path of the detected schema as a local file path for the current validated XML document.

- `/${cp}` - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- `/${tp}` - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor / Print** Preferences page.
- `/${xpath_eval(expression)}` - Evaluates an XPath 3.0 expression. Depending on the context, the expression can be:
 - *static*, when executed in a non-XML context. For example, you can use such static expressions to perform string operations on other editor variables for composing the name of the output file in a transformation scenario's **Output** tab.

Example:

```
 ${xpath_eval(upper-case(substring('${cfn}', 1, 4)))}
```

- *dynamic*, when executed in an XML context. For example, you can use such dynamic expression in a code template or as a value of an author operation's parameter.

Example:

```
 ${ask('Set new ID attribute', generic, '${xpath_eval(@id)})')}
```

- `/${i18n(key)}` - Editor variable used only at document type/framework level to allow translating names and descriptions of Author actions in multiple actions. For more details see the [Localizing Frameworks](#) on page 548 section.

Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of an external tool or of a custom validator, the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

You can configure the custom editor variables in the [Custom Editor Variables preferences page](#).

Create Your Own Stylesheet Templates

Oxygen XML Editor allows you to create your own stylesheets templates and place them in the templates directory:

- Customize the stylesheet (add namespaces etc.) that you want to become a template and save it to a file with an appropriate name.
- Copy the file to the `templates` directory in the Oxygen XML Editor installation directory.
- Open Oxygen XML Editor and go to **File > New** to see your custom template.

Configuring XML Catalogs

In the XML sample file for **SDF** you did not use a `xsi:schemaLocation` attribute, but instead you let the editor use the schema from the association. However, there are cases in which you must reference the location of a schema file from a remote web location and an Internet connection may not be available. In such cases an XML catalog may be used to map the web location to a local file system entry. The following procedure presents an example of using an XML catalogs, by modifying our `sdf.xsd` XML Schema file from the [Example Files Listings](#).

1. Create a catalog file that will help the parser locate the schema for validating the XML document. The file must map the location of the schema to a local version of the schema.

Create a new XML file called `catalog.xml` and save it into the `[OXYGEN_DIR]/frameworks/sdf` directory. The content of the file should be:

```
<?xml version="1.0"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="http://www.oxygenxml.com/SDF/abs.xsd"
```

```

        uri="schema/abs.xsd"/>
<uri name="http://www.oxygenxml.com/SDF/abs.xsd"
      uri="schema/abs.xsd"/>
</catalog>

```

2. Add catalog files to your Document Type Association using the Catalogs tab from the **Document Type** dialog box.

To test the catalog settings, restart Oxygen XML Editor and try to validate a new sample **Simple Documentation Framework** document. There should be no errors.

The sdf .xsd schema that validates the document refers the other file abs .xsd through an import element:

```

<xsi:import namespace=
  "http://www.oxygenxml.com/sample/documentation/abstracts"
  schemaLocation="http://www.oxygenxml.com/SDF/abs.xsd"/>

```

The schemaLocation attribute references the abs .xsd file:

```

xsi:schemaLocation="http://www.oxygenxml.com/sample/documentation/abstracts
  http://www.oxygenxml.com/SDF/abs.xsd"/>

```

The catalog mapping is:

```

http://www.oxygenxml.com/SDF/abs.xsd -> schema/abs.xsd

```

This means that all the references to `http://www.oxygenxml.com/SDF/abs.xsd` must be resolved to the abs .xsd file located in the schema directory. The URI element is used by URI resolvers, for example for resolving a URI reference used in an XSLT stylesheet.

Configuring Transformation Scenarios

When distributing a framework to the users, it is a good idea to have the transformation scenarios already configured. This helps the content authors publish their work in various formats. Being contained in the **Document Type Association**, the scenarios can be distributed along with the actions, menus, toolbars, and catalogs.

These are the steps that allow you to create a transformation scenario for your framework.

1. Create a `xsl` folder inside the `frameworks/sdf` folder.

The folder structure for the documentation framework should be:

```

oxygen
  frameworks
    sdf
      schema
      css
      templates
      xsl

```

2. Create the `sdf.xsl` file in the `xsl` folder. The complete content of the `sdf.xsl` file is found in the [Example Files Listings](#).

3. [Open the Preferences dialog box](#) and go to **Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework then choose the **Transformation** tab. Click the **+ New** button and choose the appropriate type of transformation (for example, **XML transformation with XSLT**).

In the **New scenario** dialog box, fill in the following fields:

- Fill in the **Name** field with *SDF to HTML*. This will be the name of your transformation scenario.
- Set the **XSL URL** field to `$(framework)/xsl/sdf.xsl`.

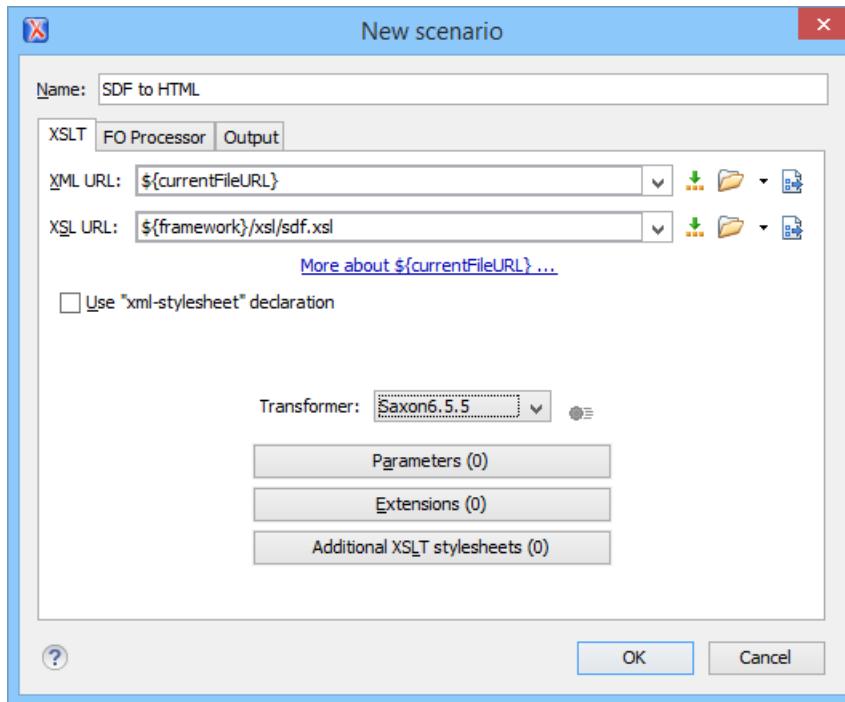


Figure 261: Configuring a New Transformation Scenario

4. Change to the **Output** tab. Configure the fields as follows:

- Set the **Save as** field to `$(cfid) / $(cfn).html`. This means the transformation output file will have the name of the XML file and the *html* extension and will be stored in the same folder.
- Enable the **Open in Browser/System Application** option.



Note: To set the browser or system application that will be used, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field. This will take precedence over the default system application settings.

- Enable the **Saved file** option.

5. Click the **OK** button to save the new scenario.

Now the scenario is listed in the **Transformation** tab:

Association rules	Schema	Classpath	Author	Templates	Catalogs	Transformation	Validation	Extensions
Type filter text								
<input type="radio"/> Default <input type="radio"/> Scenario <input checked="" type="checkbox"/> SDF to HTML								

Figure 262: The Transformation Tab

To test the transformation scenario that you just created, open the **SDF** XML sample from the [Example Files Listings](#).

Click the **Apply Transformation Scenario(s)** button to display the **Transform with** dialog box. The scenario list contains the scenario you defined earlier. Select the *SDF to HTML* scenario that you just defined and click the **Apply associated** button. The *HTML* file is saved in the same folder as the *XML* file and displayed in the browser.

Configuring Validation Scenarios

You can distribute a framework with a series of already configured validation scenarios. Also, this provides enhanced validation support that allows you to use multiple grammars to check the document. For example, you can use Schematron rules to impose guidelines, otherwise impossible to enforce using conventional validation.

To associate a validation scenario with a specific framework, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Document Type Associations**.
2. **Edit** the specific framework to open the **Document Type** dialog box, then choose the **Validation** tab. This tab displays a list of document types in which you can define validation scenarios. To set one of the validation scenarios as the default for a specific document type, check the **Default** box for that specific document type.
3. Press the  New button to add a new scenario.

The **New scenarios** dialog box that lists all validation units of the scenario is opened.

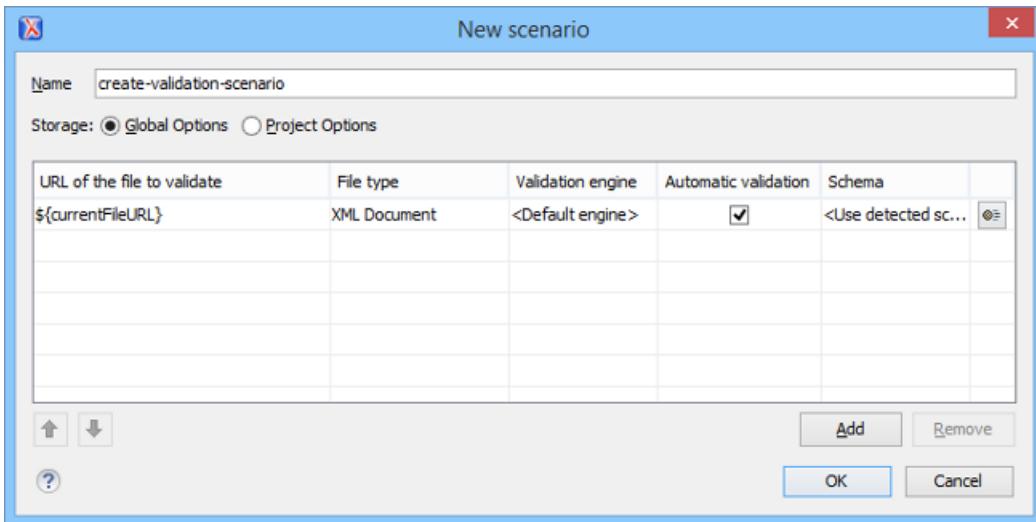


Figure 263: Add / Edit a Validation Unit

The table includes the following information:

- **URL of the file to validate** - The URL of the main module that includes the current module. It is also the entry module of the validation process when the current one is validated.
- **File type** - The type of the document that is validated in the current validation unit. Oxygen XML Editor automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Editor for validation of the type of document to which the current module belongs. **Default engine** is the default setting and it means that the default engine executes the validation. This engine is set in the **Preferences** pages for the current document type (XML document, XML Schema, XSLT stylesheet, XQuery file, etc.) instead of a validation scenario.
- **Automatic validation** - If this option is checked, the validation operation defined by this row is also applied by [the automatic validation feature](#). If the **Automatic validation** feature is [disabled in Preferences](#), then this option is ignored, as the Preference setting has a higher priority.
- **Schema** - This option becomes active when you set the **File type** to **XML Document**.
-  **Settings** - Opens the **Specify Schema** dialog box that allows you to set a schema for validating XML documents, or a list of extensions for validating XSL or XQuery documents. You can also set a default phase for validation with a Schematron schema.

4. Press the **Add** button to add a new validation unit with default settings.
5. To edit the URL of the main validation module, double-click on its cell in the **URL of the file to validate** column. Specify the URL of the main module by doing one of the following:
 - Use the  **Browse** drop-down button to browse for a local, remote, or archived file.

- Use the  **Insert Editor Variable** button to insert an *editor variable* or a *custom editor variable*.

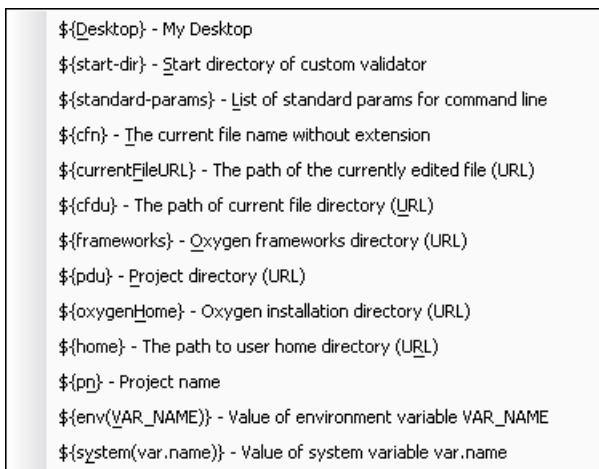


Figure 264: Insert an Editor Variable

6. Select the type of the validated document.

Note that this determines the list of possible validation engines.

7. Select the validation engine.

8. Select the **Automatic validation** option if you want to validate the current unit when the *automatic validation feature is enabled in the Preferences*.

9. Choose the schema to be used during validation (the schema detected after parsing the document or a custom one).

Configuring Extensions

You can add extensions to your Document Type Association using the **Extensions** tab from the **Document Type** dialog box.



Note: It is possible for a plugin to share the same classes with a framework. For further details, go to [How to Share the Classloader Between a Framework and a Plugin](#).

Configuring an Extensions Bundle

Starting with Oxygen XML Editor 10.3 version a single bundle was introduced acting as a provider for all other extensions. The individual extensions can still be set and if present they take precedence over the single provider, but this practice is being discouraged and the single provider should be used instead. To set individual extensions, [open the Preferences dialog box](#), go to **Document Type Association**, double-click a document type, and go to the extension tab.

The extensions bundle is represented by the `ro.sync.ecss.extensions.api.ExtensionsBundle` class. The provided implementation of the `ExtensionsBundle` is instantiated when the rules of the Document Type Association defined for the custom framework match a document opened in the editor. Therefor references to objects which need to be persistent throughout the application running session must not be kept in the bundle because the next detection event can result in creating another `ExtensionsBundle` instance.



Note: The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

1. Create a new Java project, in your IDE. Create the `lib` folder in the Java project folder and copy in it the `oxygen.jar` file from the `[OXYGEN_DIR]/lib` folder.
2. Create the class `simple.documentation.framework.SDFExtensionsBundle`, which must extend the abstract class `ro.sync.ecss.extensions.api.ExtensionsBundle`.

```
public class SDFExtensionsBundle extends ExtensionsBundle {
```

3. A **Document Type ID** and a short description should be defined first by implementing the methods `getDocumentTypeID` and `getDescription`. The Document Type ID is used to uniquely identify the current framework. Such an ID must be provided especially if options related to the framework need to be persistently stored and retrieved between sessions.

```
public String getDocumentTypeID() {
    return "Simple.Document.Framework.document.type";
}

public String getDescription() {
    return "A custom extensions bundle used for the Simple Document" +
        "Framework document type";
}
```

4. In order to be notified about the activation of the custom Author extension in relation with an opened document an `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` should be implemented. The **activation** and **deactivation** events received by this listener should be used to perform custom initializations and to register / remove listeners like `ro.sync.ecss.extensions.api.AuthorListener`, `ro.sync.ecss.extensions.api.AuthorMouseListener` or `ro.sync.ecss.extensions.api.AuthorCaretListener`. The custom author extension state listener should be provided by implementing the method `createAuthorExtensionStateListener`.

```
public AuthorExtensionStateListener createAuthorExtensionStateListener() {
    return new SDFAuthorExtensionStateListener();
}
```

The `AuthorExtensionStateListener` is instantiated and notified about the activation of the framework when the rules of the Document Type Association match a document opened in the Author editor mode. The listener is notified about the deactivation when another framework is activated for the same document, the user switches to another mode or the editor is closed. A complete description and implementation of an `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` can be found in the [Implementing an Author Extension State Listener](#).

If Schema Aware mode is active in Oxygen XML Editor, all actions that can generate invalid content will be redirected toward the `ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler`. The handler can either resolve a specific case, let the default implementation take place or reject the edit entirely by throwing an `ro.sync.ecss.extensions.api.InvalidEditException`. The actions that are forwarded to this handler include typing, delete or paste.

See [Implementing an Author Schema Aware Editing Handler](#) on page 570 for more details about this handler.

5. Customizations of the content completion proposals are permitted by creating a schema manager filter extension. The interface that declares the methods used for content completion proposals filtering is `ro.sync.contentcompletion.xml.SchemaManagerFilter`. The filter can be applied on elements, attributes or on their values. Responsible for creating the content completion filter is the method `createSchemaManagerFilter`. A new `SchemaManagerFilter` will be created each time a document matches the rules defined by the Document Type Association which contains the filter declaration.

```
public SchemaManagerFilter createSchemaManagerFilter() {
    return new SDFSchemaManagerFilter();
}
```

A detailed presentation of the schema manager filter can be found in [Configuring a Content completion handler](#) section.

6. The Author supports link based navigation between documents and document sections. Therefore, if the document contains elements defined as links to other elements, for example links based on the `id` attributes, the extension should provide the means to find the referenced content. To do this an implementation of the `ro.sync.ecss.extensions.api.link.ElementLocatorProvider` interface should be returned by

the `createElementLocatorProvider` method. Each time an element pointed by a link needs to be located the method is invoked.

```
public ElementLocatorProvider createElementLocatorProvider() {
    return new DefaultElementLocatorProvider();
}
```

The section that explains how to implement an element locator provider is [Configuring a Link target element finder](#).

7. The drag and drop functionality can be extended by implementing the `ro.sync.exml.editor.xmleditor.pageauthor.AuthorDnDListener` interface. Relevant methods from the listener are invoked when the mouse is dragged, moved over, or exits the Author editor mode, when the drop action changes, and when the drop occurs. Each method receives the `DropTargetEvent` containing information about the drag and drop operation. The drag and drop extensions are available on Author mode for both Oxygen XML Editor Eclipse plugin and standalone application. The Text mode corresponding listener is available only for Oxygen XML Editor Eclipse plugin. The methods corresponding to each implementation are: `createAuthorAWTDndListener`, `createTextSWTDndListener` and `createAuthorSWTDndListener`.

```
public AuthorDnDListener createAuthorAWTDndListener() {
    return new SDFAuthorDndListener();
}
```

For more details about the Author drag and drop listeners see the [Configuring a custom Drag and Drop listener](#) section.

8. Another extension which can be included in the bundle is the reference resolver. In our case the references are represented by the `ref` element and the attribute indicating the referenced resource is **location**. To be able to obtain the content of the referenced resources you will have to implement a Java extension class which implements the `ro.sync.ecss.extensions.api.AuthorReferenceResolver`. The method responsible for creating the custom references resolver is `createAuthorReferenceResolver`. The method is called each time a document opened in an Author editor mode matches the Document Type Association where the extensions bundle is defined. The instantiated references resolver object is kept and used until another extensions bundle corresponding to another Document Type is activated as result of the detection process.

```
public AuthorReferenceResolver createAuthorReferenceResolver() {
    return new ReferencesResolver();
}
```

A more detailed description of the references resolver can be found in the [Configuring a References Resolver](#) section.

9. To be able to dynamically customize the default CSS styles for a certain `ro.sync.ecss.extensions.api.node.AuthorNode` an implementation of the `ro.sync.ecss.extensions.api.StylesFilter` can be provided. The extensions bundle method responsible for creating the `StylesFilter` is `createAuthorStylesFilter`. The method is called each time a document opened in an Author editor mode matches the document type association where the extensions bundle is defined. The instantiated filter object is kept and used until another extensions bundle corresponding to another Document Type is activated as a result of the detection process.

```
public StylesFilter createAuthorStylesFilter() {
    return new SDFStylesFilter();
}
```

See the [Configuring CSS styles filter](#) section for more details about the styles filter extension.

10. In order to edit data in custom tabular format implementations of the `ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider` and `the ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` interfaces should be provided.

The two methods from the `ExtensionsBundle` specifying these two extension points are `createAuthorTableCellSpanProvider` and `createAuthorTableColumnWidthProvider`.

```
public AuthorTableCellSpanProvider createAuthorTableCellSpanProvider() {
    return new TableCellSpanProvider();
}

public AuthorTableColumnWidthProvider
        createAuthorTableColumnWidthProvider() {
    return new TableColumnWidthProvider();
}
```

The two table information providers are not reused for different tables. The methods are called for each table in the document so new instances should be provided every time. Read more about the cell span and column width information providers in [Configuring a Table Cell Span Provider](#) and [Configuring a Table Column Width Provider](#) sections.

If the functionality related to one of the previous extension point does not need to be modified then the developed `ro.sync.ecss.extensions.api.ExtensionsBundle` should not override the corresponding method and leave the default base implementation to return `null`.

11. An XML vocabulary can contain links to different areas of a document. In case the document contains elements defined as link you can choose to present a more relevant text description for each link. To do this an implementation of the `ro.sync.ecss.extensions.api.link.LinkTextResolver` interface should be returned by the `createLinkTextResolver` method. This implementation is used each time `the oxy_link-text() function` is encountered in the CSS styles associated with an element.

```
public LinkTextResolver createLinkTextResolver() {
    return new DitaLinkTextResolver();
}
```

Oxygen XML Editor offers built in implementations for DITA and DocBook:

`ro.sync.ecss.extensions.dita.link.DitaLinkTextResolver`
`ro.sync.ecss.extensions.docbook.link.DocbookLinkTextResolver`

12. Pack the compiled class into a jar file.
13. Copy the jar file into the `frameworks/sdf` directory.
14. Add the jar file to the Author class path.
15. Register the Java class by going to the **Extensions** tab. Press the **Choose** button and select the name of the class: `SDFExtensionsBundle`.

 **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the `oxygen-sample-framework` module of the [Oxygen SDK](#), available as a Maven archetype [on the Oxygen XML Editor website](#).

Customize Profiling Conditions

For each document type, you can configure the phrase-type elements that wrap the profiled content by setting a custom `ro.sync.ecss.extensions.api.ProfilingConditionalTextProvider`. This configuration is set by default for DITA and DocBook frameworks.

Customizing Smart Paste Support

The `smart paste` functionalist preserves certain style and structure information when pasting to certain `document types` from certain common applications.

For other document types the default behavior of the paste operation is to keep only the text content without the styling but it can be customized by setting an XSLT stylesheet in that document type. The XSLT stylesheet must accept as input an XHTML flavor of the copied content and transform it to the equivalent XML markup that is appropriate for the target document type of the paste operation. The stylesheet is [set up](#) by implementing the `getImporterStylesheetFileName` method of an instance object of `the AuthorExternalObjectInsertionHandler class` which is returned by the

`createExternalObjectInsertionHandler` method of *the ExtensionsBundle instance* of the target document type.

Implementing an Author Extension State Listener

The `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` implementation is notified when the Author extension where the listener is defined is activated or deactivated in the Document Type detection process.

 **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Editor website*. Also it is available in the *Author SDK Maven Project*.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorExtensionStateListener;

public class SDFAuthorExtensionStateListener implements
    AuthorExtensionStateListener {
    private AuthorListener sdfAuthorDocumentListener;
    private AuthorMouseListener sdfMouseListener;
    private AuthorCaretListener sdfCaretListener;
    private OptionListener sdfOptionListener;
```

The **activation** event received by this listener when the rules of the Document Type Association match a document opened in the Author editor mode, should be used to perform custom initializations and to register listeners like `ro.sync.ecss.extensions.api.AuthorListener`, `ro.sync.ecss.extensions.api.AuthorMouseListener` or `ro.sync.ecss.extensions.api.AuthorCaretListener`.

```
public void activated(AuthorAccess authorAccess) {
    // Get the value of the option.
    String option = authorAccess.getOptionsStorage().getOption(
        "sdf.custom.option.key", "");
    // Use the option for some initializations...

    // Add an option listener.
    authorAccess.getOptionsStorage().addOptionListener(sdfOptionListener);

    // Add author document listeners.
    sdfAuthorDocumentListener = new SDFAuthorListener();
    authorAccess.getDocumentController().addAuthorListener(
        sdfAuthorDocumentListener);

    // Add mouse listener.
    sdfMouseListener = new SDFAuthorMouseListener();
    authorAccess.getEditorAccess().addAuthorMouseListener(sdfMouseListener);

    // Add caret listener.
    sdfCaretListener = new SDFAuthorCaretListener();
    authorAccess.getEditorAccess().addAuthorCaretListener(sdfCaretListener);

    // Other custom initializations...
}
```

The `authorAccess` parameter received by the `activated` method can be used to gain access to Author specific actions and informations related to components like the editor, document, workspace, tables, or the change tracking manager.

If options specific to the custom developed Author extension need to be stored or retrieved, a reference to the `ro.sync.ecss.extensions.api.OptionsStorage` can be obtained by calling the `getOptionsStorage` method from the author access. The same object can be used to register `ro.sync.ecss.extensions.api.OptionListener` listeners. An option listener is registered in relation with an option **key** and will be notified about the value changes of that option.

An `AuthorListener` can be used if events related to the Author document modifications are of interest. The listener can be added to the `ro.sync.ecss.extensions.api.AuthorDocumentController`. A reference to the document controller is returned by the `getDocumentController` method from the author access. The document controller can also be used to perform operations involving document modifications.

To provide access to Author editor component related functionality and information, the author access has a reference to the `ro.sync.ecss.extensions.api.access.AuthorEditorAccess` that can be obtained when calling

the `getEditorAccess` method. At this level `AuthorMouseListener` and `AuthorCaretListener` can be added which will be notified about mouse and caret events occurring in the Author editor mode.

The **deactivation** event is received when another framework is activated for the same document, the user switches to another editor mode or the editor is closed. The `deactivate` method is typically used to unregister the listeners previously added on the `activate` method and to perform other actions. For example, options related to the deactivated author extension can be saved at this point.

```
public void deactivated(AuthorAccess authorAccess) {
    // Store the option.
    authorAccess.getOptionsStorage().setOption(
        "sdf.custom.option.key", optionValue);

    // Remove the option listener.
    authorAccess.getOptionsStorage().removeOptionListener(sdfOptionListener);

    // Remove document listeners.
    authorAccess.getDocumentController().removeAuthorListener(
        sdfAuthorDocumentListener);

    // Remove mouse listener.
    authorAccess.getEditorAccess().removeAuthorMouseListener(sdfMouseListener);

    // Remove caret listener.
    authorAccess.getEditorAccess().removeAuthorCaretListener(sdfCaretListener);

    // Other actions...
}
```

Implementing an Author Schema Aware Editing Handler

To implement your own handler for actions like typing, deleting, or pasting, provide an implementation of `ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler`. For this handler to be called, the **Schema Aware Editing option** must be set to **On**, or **Custom**. The handler can either resolve a specific case, let the default implementation take place, or reject the edit entirely by throwing an `InvalidEditException`.

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

```
package simple.documentation.framework.extensions;

/**
 * Specific editing support for SDF documents.
 * Handles typing and paste events inside section and tables.
 */
public class SDFSchemaAwareEditingHandler implements AuthorSchemaAwareEditingHandler {
```

Typing events can be handled using the `handleTyping` method. For example, the `SDFSchemaAwareEditingHandler` checks if the schema is not a learned one, was loaded successfully and **Smart Paste** is active. If these conditions are met, the event will be handled.

```
/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleTyping(int, char,
 * ro.sync.ecss.extensions.api.AuthorAccess)
 */
public boolean handleTyping(int offset, char ch, AuthorAccess authorAccess)
throws InvalidEditException {
    boolean handleTyping = false;
    AuthorSchemaManager authorSchemaManager = authorAccess.getDocumentController().getAuthorSchemaManager();
    if (!authorSchemaManager.isLearnSchema() &&
        !authorSchemaManager.hasLoadingErrors() &&
        authorSchemaManager.getAuthorSchemaAwareOptions().isEnableSmartTyping()) {
        try {
            AuthorDocumentFragment characterFragment =
                authorAccess.getDocumentController().createNewDocumentTextFragment(String.valueOf(ch));
            handleTyping = handleInsertionEvent(offset, new AuthorDocumentFragment[] {characterFragment}, authorAccess);
        } catch (AuthorOperationException e) {
            throw new InvalidEditException(e.getMessage(), "Invalid typing event: " + e.getMessage(), e, false);
        }
    }
    return handleTyping;
}
```

Implementing the `AuthorSchemaAwareEditingHandler` gives the possibility to handle other events like: the keyboard delete event at the given offset (using Delete or Backspace keys), delete element tags, delete selection, join elements or paste fragment.



Note: The complete source code can be found in the Simple Documentation Framework project, included in the `oxygen-sample-framework` module of the *Oxygen SDK*, available as a Maven archetype [on the Oxygen XML Editor website](#).

Configuring a Content Completion Handler

You can filter or contribute to items offered for content completion by implementing the `ro.sync.contentcompletion.xml.SchemaManagerFilter` interface.



Note: The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the *Oxygen SDK Maven Project*.

```
import java.util.List;
import ro.sync.contentcompletion.xml.CIAtribute;
import ro.sync.contentcompletion.xml.CIElement;
import ro.sync.contentcompletion.xml.CIValue;
import ro.sync.contentcompletion.xml.Context;
import ro.sync.contentcompletion.xml.SchemaManagerFilter;
import ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatElementsCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatPossibleValuesHasAttributeContext;

public class SDFSchemaManagerFilter implements SchemaManagerFilter {
```

You can implement the various callbacks of the interface either by returning the default values given by Oxygen XML Editor or by contributing to the list of proposals. The filter can be applied on elements, attributes or on their values. Attributes filtering can be implemented using the `filterAttributes` method and changing the default content completion list of `ro.sync.contentcompletion.xml.CIAtribute` for the element provided by the current `ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext` context. For example, the `SDFSchemaManagerFilter` checks if the element from the current context is the `table` element and adds the `frame` attribute to the `table` list of attributes.

```
/**
 * Filter attributes of the "table" element.
 */
public List<CIAtribute> filterAttributes(List<CIAtribute> attributes,
    WhatAttributesCanGoHereContext context) {
    // If the element from the current context is the 'table' element add the
    // attribute named 'frame' to the list of default content completion proposals
    if (context != null) {
        ContextElement contextElement = context.getParentElement();
        if ("table".equals(contextElement.getQName())) {
            CIAtribute frameAttribute = new CIAtribute();
            frameAttribute.setName("frame");
            frameAttribute.setRequired(false);
            frameAttribute.setFixed(false);
            frameAttribute.setDefaultValue("void");
            if (attributes == null) {
                attributes = new ArrayList<CIAtribute>();
            }
            attributes.add(frameAttribute);
        }
    }
    return attributes;
}
```

The elements that can be inserted in a specific context can be filtered using the `filterElements` method. The `SDFSchemaManagerFilter` uses this method to replace the `td` child element with the `th` element when `header` is the current context element.

```
public List<CIElement> filterElements(List<CIElement> elements,
    WhatElementsCanGoHereContext context) {
    // If the element from the current context is the 'header' element remove the
    // 'td' element from the list of content completion proposals and add the
    // 'th' element.
    if (context != null) {
        Stack<ContextElement> elementStack = context.getElementStack();
        if (elementStack != null) {
```

```

ContextElement contextElement = context.getElementStack().peek();
if ("header".equals(contextElement.getQName())) {
    if (elements != null) {
        for (Iterator<CIElement> iterator = elements.iterator(); iterator.hasNext();) {
            CIElement element = iterator.next();
            // Remove the 'td' element
            if ("td".equals(element.getQName())) {
                elements.remove(element);
                break;
            }
        }
    } else {
        elements = new ArrayList<CIElement>();
    }
    // Insert the 'th' element in the list of content completion proposals
    CIElement thElement = new SDFElement();
    thElement.setName("th");
    elements.add(thElement);
}
} else {
    // If the given context is null then the given list of content completion elements contains
    // global elements.
}
return elements;
}

```

The elements or attributes values can be filtered using the `filterElementValues` or `filterAttributeValues` methods.



Note: The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK*, available as a Maven archetype [on the Oxygen XML Editor website](#).

Configuring a Link target element finder

The link target reference finder represents the support for finding references from links which indicate specific elements inside an XML document. This support will only be available if a schema is associated with the document type.

If you do not define a custom link target reference finder, the `DefaultElementLocatorProvider` implementation will be used by default. The interface which should be implemented for a custom link target reference finder is `ro.sync.ecss.extensions.api.link.ElementLocatorProvider`. As an alternative, the `ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider` implementation can also be extended.

The used `ElementLocatorProvider` will be queried for an `ElementLocator` when a link location must be determined (when a link is clicked). Then, to find the corresponding (linked) element, the obtained `ElementLocator` will be queried for each element from the document.



Note: The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

The DefaultElementLocatorProvider implementation

The `DefaultElementLocatorProvider` implementation offers support for the most common types of links:

- links based on ID attribute values
- XPointer element() scheme

The method `getElementLocator` determines what `ElementLocator` should be used. In the default implementation it checks if the link is an XPointer element() scheme otherwise it assumes it is an ID. A non-null `IDTypeVerifier` will always be provided if a schema is associated with the document type.

The link string argument is the "anchor" part of the URL which is composed from the value of the link property specified for the link element in the CSS.

```

public ElementLocator getElementLocator(IDTypeVerifier idVerifier,
                                         String link) {
    ElementLocator elementLocator = null;
    try {

```

```

    if(link.startsWith("element")){
        // xpointer element() scheme
        elementLocator = new XPointerElementLocator(idVerifier, link);
    } else {
        // Locate link element by ID
        elementLocator = new IDEElementLocator(idVerifier, link);
    }
} catch (ElementLocatorException e) {
    logger.warn("Exception when create element locator for link: "
            + link + ". Cause: " + e, e);
}
return elementLocator;
}

```

The XPointerElementLocator implementation

XPointerElementLocator is an implementation of the abstract class

[ro.sync.ecss.extensions.api.link.ElementLocator](#) for links that have one of the following XPointer element() scheme patterns:

element(elementID)

Locate the element with the specified id.

element(/1/2/3)

A child sequence appearing alone identifies an element by means of stepwise navigation, which is directed by a sequence of integers separated by slashes (/); each integer n locates the nth child element of the previously located element.

element(elementID/3/4)

A child sequence appearing after a *NCName* identifies an element by means of stepwise navigation, starting from the element located by the given name.

The constructor separates the id/integers which are delimited by slashes(/) into a sequence of identifiers (an XPointer path). It will also check that the link has one of the supported patterns of the XPointer element() scheme.

```

public XPointerElementLocator(IDTypeVerifier idVerifier, String link)
    throws ElementLocatorException {
super(link);
this.idVerifier = idVerifier;

link = link.substring("element".length(), link.length() - 1);

StringTokenizer stringTokenizer = new StringTokenizer(link, "/", false);
xpointerPath = new String[stringTokenizer.countTokens()];
int i = 0;
while (stringTokenizer.hasMoreTokens()) {
    xpointerPath[i] = stringTokenizer.nextToken();
    boolean invalidFormat = false;

    // Empty xpointer component is not supported
    if(xpointerPath[i].length() == 0){
        invalidFormat = true;
    }

    if(i > 0){
        try {
            Integer.parseInt(xpointerPath[i]);
        } catch (NumberFormatException e) {
            invalidFormat = true;
        }
    }

    if(invalidFormat){
        throw new ElementLocatorException(
            "Only the element() scheme is supported when locating XPointer links."
            + "Supported formats: element(elementID), element(/1/2/3),"
            + "element(elementID/2/3/4).");
    }
    i++;
}

if(Character.isDigit(xpointerPath[0].charAt(0))){
    // This is the case when xpointer have the following pattern /1/5/
    xpointerPathDepth = xpointerPath.length;
} else {
    // This is the case when xpointer starts with an element ID
    xpointerPathDepth = -1;
    startWithElementID = true;
}

```

```
    }
```

The method `startElement` will be invoked at the beginning of every element in the XML document(even when the element is empty). The arguments it takes are

uri

The namespace URI, or the empty string if the element has no namespace URI or if namespace processing is disabled.

localName

Local name of the element.

qName

Qualified name of the element.

atts

Attributes attached to the element. If there are no attributes, this argument will be empty.

The method returns `true` if the processed element is found to be the one indicated by the link.

The `XPointerElementLocator` implementation of the `startElement` will update the depth of the current element and keep the index of the element in its parent. If the `xpointerPath` starts with an element ID then the current element ID is verified to match the specified ID. If this is the case the depth of the XPointer is updated taking into account the depth of the current element.

If the XPointer path depth is the same as the current element depth then the kept indices of the current element path are compared to the indices in the XPointer path. If all of them match then the element has been found.

```

public boolean startElement(String uri, String localName,
    String name, Attr[] attrs) {
    boolean linkLocated = false;
    // Increase current element document depth
    startElementDepth++;

    if (endElementDepth != startElementDepth) {
        // The current element is the first child of the parent
        currentElementIndexStack.push(new Integer(1));
    } else {
        // Another element in the parent element
        currentElementIndexStack.push(new Integer(lastIndexInParent + 1));
    }

    if (startsWithElementID) {
        // This the case when xpointer path starts with an element ID.
        String xpointerElement = xpointerPath[0];
        for (int i = 0; i < attrs.length; i++) {
            if(xpointerElement.equals(attrs[i].getValue())){
                if(idVerifier.hasIDType(
                    localName, uri, attrs[i].getQName(), attrs[i].getNamespace())){
                    xpointerPathDepth = startElementDepth + xpointerPath.length - 1;
                    break;
                }
            }
        }
    }

    if (xpointerPathDepth == startElementDepth){
        // check if xpointer path matches with the current element path
        linkLocated = true;
        try {
            int xpointerIdx = xpointerPath.length - 1;
            int stackIdx = currentElementIndexStack.size() - 1;
            int stopIdx = startWithElementID ? 1 : 0;
            while (xpointerIdx >= stopIdx && stackIdx >= 0) {
                int xpointerIndex = Integer.parseInt(xpointerPath[xpointerIdx]);
                int currentElementIndex =
                    ((Integer)currentElementIndexStack.get(stackIdx)).intValue();
                if(xpointerIndex != currentElementIndex) {
                    linkLocated = false;
                    break;
                }

                xpointerIdx--;
                stackIdx--;
            }
        }
    }
}

```

```

        } catch (NumberFormatException e) {
            logger.warn(e,e);
        }
    }
    return linkLocated;
}

```

The method `endElement` will be invoked at the end of every element in the XML document (even when the element is empty).

The `XPointerElementLocator` implementation of the `endElement` updates the depth of the current element path and the index of the element in its parent.

```

public void endElement(String uri, String localName, String name) {
    endElementDepth = startElementDepth;
    startElementDepth--;
    lastIndexInParent = ((Integer)currentElementIndexStack.pop()).intValue();
}

```

The `IDElementLocator` implementation

The `IDElementLocator` is an implementation of the abstract class [ro.sync.ecss.extensions.api.link.ElementLocator](#) for links that use an **id**.

The constructor only assigns field values and the method `endElement` is empty for this implementation.

The method `startElement` checks each of the element's attribute values and when one matches the link, it considers the element found if one of the following conditions is satisfied:

- the qualified name of the attribute is `xml:id`
- the attribute type is ID

The attribute type is checked with the help of the method `IDTypeVerifier.hasIDType`.

```

public boolean startElement(String uri, String localName,
                           String name, Attr[] attrs) {
    boolean elementFound = false;
    for (int i = 0; i < attrs.length; i++) {
        if (link.equals(attrs[i].getValue())) {
            if ("xml:id".equals(attrs[i].getQName())) {
                // xml:id attribute
                elementFound = true;
            } else {
                // check if attribute has ID type
                String attrLocalName =
                    ExtensionUtil.getLocalName(attrs[i].getQName());
                String attrUri = attrs[i].getNamespace();
                if (idVerifier.hasIDType(localName, uri, attrLocalName, attrUri)) {
                    elementFound = true;
                }
            }
        }
    }
    return elementFound;
}

```

Creating a customized link target reference finder

If you need to create a custom link target reference finder you can do so by creating the class which will implement the [ro.sync.ecss.extensions.api.link.ElementLocatorProvider](#) interface. As an alternative, your class could extend [ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider](#), the default implementation.



Note: The complete source code of the

[ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider](#),
[ro.sync.ecss.extensions.commons.IDElementLocator](#) or
[ro.sync.ecss.extensions.commons.XPointerElementLocator](#) can be found in the
[oxygen-sample-framework](#) project.

Configuring a custom Drag and Drop listener

Sometimes it is useful to perform various operations when certain objects are dropped from outside sources in the editing area. You can choose from three interfaces to implement depending on whether you are using the framework with the Eclipse plugin or the standalone version of the application or if you want to add the handler for the Text or Author modes.

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

Table 9: Interfaces for the Drag and Drop listener

Interface	Description
<code>ro.sync.xml.editor.xmleditor.pageauthor.AuthorDnDListener</code>	Receives callbacks from the standalone application for Drag And Drop in Author mode.
<code>com.oxygenxml.editor.editors.author.AuthorDnDListener</code>	Receives callbacks from the Eclipse plugin for Drag And Drop in Author mode.
<code>com.oxygenxml.editor.editors.TextDnDListener</code>	Receives callbacks from the Eclipse plugin for Drag And Drop in Text mode.

Configuring a References Resolver

You need to provide a handler for resolving references and obtain the content they reference. In our case the element which has references is **ref** and the attribute indicating the referenced resource is **location**. You will have to implement a Java extension class for obtaining the referenced resources.

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

1. Create the class `simple.documentation.framework REFERENCESResolver`. This class must implement the `ro.sync.ecss.extensions.api.AuthorReferenceResolver` interface.

```
import ro.sync.ecss.extensions.api.AuthorReferenceResolver;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;
import ro.sync.ecss.extensions.api.node.AuthorNode;

public class ReferencesResolver
    implements AuthorReferenceResolver {
```

2. The `hasReferences` method verifies if the handler considers the node to have references. It takes as argument an `AuthorNode` that represents the node which will be verified. The method will return `true` if the node is considered to have references. In our case, to be a reference the node must be an element with the name `ref` and it must have an attribute named `location`.

```
public boolean hasReferences(AuthorNode node) {
    boolean hasReferences = false;
    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
        AuthorElement element = (AuthorElement) node;
        if ("ref".equals(element.getLocalName())) {
            AttrValue attrValue = element.getAttribute("location");
            hasReferences = attrValue != null;
        }
    }
    return hasReferences;
}
```

3. The method `getDisplayName` returns the display name of the node that contains the expanded referenced content. It takes as argument an `AuthorNode` that represents the node for which the display name is needed. The referenced content engine will ask this `AuthorReferenceResolver` implementation what is the display name for each

node which is considered a reference. In our case the display name is the value of the *location* attribute from the *ref* element.

```
public String getDisplayName(AuthorNode node) {
    String displayName = "ref-fragment";
    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
        AuthorElement element = (AuthorElement) node;
        if ("ref".equals(element.getLocalName())) {
            AttrValue attrValue = element.getAttribute("location");
            if (attrValue != null) {
                displayName = attrValue.getValue();
            }
        }
    }
    return displayName;
}
```

- The method `resolveReference` resolves the reference of the node and returns a `SAXSource` with the parser and the parser's input source. It takes as arguments an `AuthorNode` that represents the node for which the reference needs resolving, the `systemID` of the node, the `AuthorAccess` with access methods to the Author data model and a `SAX EntityResolver` which resolves resources that are already opened in another editor or resolve resources through the XML catalog. In the implementation you need to resolve the reference relative to the `systemID`, and create a parser and an input source over the resolved reference.

```
public SAXSource resolveReference(
    AuthorNode node,
    String systemID,
    AuthorAccess authorAccess,
    EntityResolver entityResolver) {
    SAXSource saxSource = null;

    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
        AuthorElement element = (AuthorElement) node;
        if ("ref".equals(element.getLocalName())) {
            AttrValue attrValue = element.getAttribute("location");
            if (attrValue != null) {
                String attrStringVal = attrValue.getValue();
                try {
                    URL absoluteUrl = new URL(new URL(systemID),
                        authorAccess.getUtilAccess().correctURL(attrStringVal));

                    InputSource inputSource = entityResolver.resolveEntity(null,
                        absoluteUrl.toString());
                    if (inputSource == null) {
                        inputSource = new InputSource(absoluteUrl.toString());
                    }
                }
                XMLReader xmlReader = authorAccess.newNonValidatingXMLReader();
                xmlReader.setEntityResolver(entityResolver);

                saxSource = new SAXSource(xmlReader, inputSource);
            } catch (MalformedURLException e) {
                logger.error(e, e);
            } catch (SAXException e) {
                logger.error(e, e);
            } catch (IOException e) {
                logger.error(e, e);
            }
        }
    }
    return saxSource;
}
```

- The method `getReferenceUniqueID` should return an unique identifier for the node reference. The unique identifier is used to avoid resolving the references recursively. The method takes as argument an `AuthorNode` that represents the node with the reference. In the implementation the unique identifier is the value of the *location* attribute from the *ref* element.

```
public String getReferenceUniqueID(AuthorNode node) {
    String id = null;
    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
        AuthorElement element = (AuthorElement) node;
        if ("ref".equals(element.getLocalName())) {
            AttrValue attrValue = element.getAttribute("location");
            if (attrValue != null) {
                id = attrValue.getValue();
            }
        }
    }
}
```

```
}
```

6. The method `getReferenceSystemID` should return the *systemID* of the referenced content. It takes as arguments an `AuthorNode` that represents the node with the reference and the `AuthorAccess` with access methods to the Author data model. In the implementation you use the value of the *location* attribute from the *ref* element and resolve it relatively to the XML base URL of the node.

```

public String getReferenceSystemID(AuthorNode node,
                                   AuthorAccess authorAccess) {
    String systemID = null;
    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
        AuthorElement element = (AuthorElement) node;
        if ("ref".equals(element.getLocalName())) {
            AttrValue attrValue = element.getAttribute("location");
            if (attrValue != null) {
                String attrStringVal = attrValue.getValue();
                try {
                    URL absoluteUrl = new URL(node.getXMLBaseURL(),
                        authorAccess.getUtilAccess().correctURL(attrStringVal));
                    systemID = absoluteUrl.toString();
                } catch (MalformedURLException e) {
                    logger.error(e, e);
                }
            }
        }
    }
    return systemID;
}

```



Note: The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the [Oxygen SDK](#), available as a Maven archetype [on the Oxygen XML Editor website](#).

In the listing below, the XML document contains the **ref** element:

```
<ref location="referenced.xml">Reference</ref>
```

When no reference resolver is specified, the reference has the following layout:

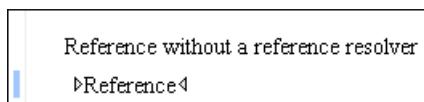


Figure 265: Reference with no specified reference resolver

When the above implementation is configured, the reference has the expected layout:

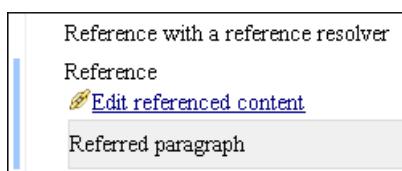


Figure 266: Reference with reference resolver

Configuring CSS Styles Filter

You can modify the CSS styles for each `ro.sync.ecss.extensions.api.node.AuthorNode` rendered in the Author mode using an implementation of `ro.sync.ecss.extensions.api.StylesFilter`. You can implement the various callbacks of the interface either by returning the default value given by Oxygen XML Editor or by contributing to the value. The received styles `ro.sync.ecss.css.Styles` can be processed and values can be overwritten with your own. For example you can override the KEY_BACKGROUND_COLOR style to return your own implementation of `ro.sync.exml.view.graphics.Color` or override the KEY_FONT style to return your own implementation of `ro.sync.exml.view.graphics.Font`.



Note: The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

For instance in our simple document example the filter can change the value of the KEY_FONT property for the table element:

```
package simple.documentation.framework;

import ro.sync.ecss.css.Styles;
import ro.sync.ecss.extensions.api.StylesFilter;
import ro.sync.ecss.extensions.api.node.AuthorNode;
import ro.sync.exml.view.graphics.Font;

public class SDFStylesFilter implements StylesFilter {

    public Styles filter(Styles styles, AuthorNode authorNode) {
        if (AuthorNode.NODE_TYPE_ELEMENT == authorNode.getType()
            && "table".equals(authorNode.getName())) {
            styles.setProperty(Styles.KEY_FONT, new Font(null, Font.BOLD, 12));
        }
        return styles;
    }
}
```

Configuring tables

There are standard CSS properties used to indicate what elements are tables, table rows and table cells. What CSS is missing is the possibility to indicate the cell spanning, row separators or the column widths. Oxygen XML Editor Author offers support for adding extensions to solve these problems. This will be presented in the next chapters.

The table in this example is a simple one. The header must be formatted in a different way than the ordinary rows, so it will have a background color.

```
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
    width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
    display:table-cell;
    border:1px solid navy;
    padding:1em;
}
```

Because in the [schema](#) the td tag has the attributes **row_span** and **column_span** that are not automatically recognized by Oxygen XML Editor Author, a Java extension will be implemented which will provide information about the cell spanning. See the section [Configuring a Table Cell Span Provider](#).

The column widths are specified by the attributes **width** of the elements **customcol** that are not automatically recognized by Oxygen XML Editor Author. It is necessary to implement a Java extension which will provide information about the column widths. See the section [Configuring a Table Column Width Provider](#).

The table from our example does not make use of the attributes **colsep** and **rowsep** (which are automatically recognized) but we still want the rows to be separated by horizontal lines. It is necessary to implement a Java extension which will provide information about the row and column separators. See the section [Configuring a Table Cell Row And Column Separator Provider](#) on page 585.

Configuring a Table Column Width Provider

In the sample documentation framework the `table` element as well as the table columns can have specified widths. In order for these widths to be considered by Author we need to provide the means to determine them. As explained in the [Configuring tables](#) on page 579, if you use the `table` element attribute `width` Oxygen XML Editor can determine the table width automatically. In this example the table has `col` elements with `width` attributes that are not recognized by default. You will need to implement a Java extension class to determine the column widths.

-  **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

1. Create the class `simple.documentation.framework.TableColumnWidthProvider`. This class must implement the `ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` interface.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperationException;
import ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider;
import ro.sync.ecss.extensions.api.WidthRepresentation;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableColumnWidthProvider
    implements AuthorTableColumnWidthProvider {
```

2. Method `init` is taking as argument an `ro.sync.ecss.extensions.api.node.AuthorElement` that represents the XML `table` element. In our case the column widths are specified in `col` elements from the `table` element. In such cases you must collect the span information by analyzing the `table` element.

```
public void init(AuthorElement tableElement) {
    this.tableElement = tableElement;
    AuthorElement[] colChildren = tableElement.getElementsByLocalName("customcol");
    if (colChildren != null && colChildren.length > 0) {
        for (int i = 0; i < colChildren.length; i++) {
            AuthorElement colChild = colChildren[i];
            if (i == 0) {
                colsStartOffset = colChild.getStartOffset();
            }
            if (i == colChildren.length - 1) {
                colsEndOffset = colChild.getEndOffset();
            }
            // Determine the 'width' for this col.
            AttrValue colWidthAttribute = colChild.getAttribute("width");
            String colWidth = null;
            if (colWidthAttribute != null) {
                colWidth = colWidthAttribute.getValue();
                // Add WidthRepresentation objects for the columns this 'customcol' specification
                // spans over.
                colWidthSpecs.add(new WidthRepresentation(colWidth, true));
            }
        }
    }
}
```

3. The method `isTableAcceptingWidth` should check if the table cells are `td`.

```
public boolean isTableAcceptingWidth(String tableCellsTagName) {
    return "td".equals(tableCellsTagName);
}
```

4. The method `isTableAndColumnsResizable` should check if the table cells are `td`. This method determines if the table and its columns can be resized by dragging the edge of a column.

```
public boolean isTableAndColumnsResizable(String tableCellsTagName) {
    return "td".equals(tableCellsTagName);
}
```

5. Methods `getTableWidth` and `getCellWidth` are used to determine the table and column width. The table layout engine will ask this `ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` implementation what is the table width for each table element and the cell width for each cell element from the table that was marked as cell in the CSS using the property `display:table-cell`. The implementation is simple and

just parses the value of the **width** attribute. The methods must return null for the tables / cells that do not have a specified width.

```
public WidthRepresentation getTableWidth(String tableCellsTagName) {
    WidthRepresentation toReturn = null;
    if (tableElement != null && "td".equals(tableCellsTagName)) {
        AttrValue widthAttr = tableElement.getAttribute("width");
        if (widthAttr != null) {
            String width = widthAttr.getValue();
            if (width != null) {
                toReturn = new WidthRepresentation(width, true);
            }
        }
    }
    return toReturn;
}

public List<WidthRepresentation> getCellWidth(AuthorElement cellElement, int colNumberStart,
    int colSpan) {
    List<WidthRepresentation> toReturn = null;
    int size = colWidthSpecs.size();
    if (size >= colNumberStart && size >= colNumberStart + colSpan) {
        toReturn = new ArrayList<WidthRepresentation>(colSpan);
        for (int i = colNumberStart; i < colNumberStart + colSpan; i++) {
            // Add the column widths
            toReturn.add(colWidthSpecs.get(i));
        }
    }
    return toReturn;
}
```

- Methods `commitTableWidthModification` and `commitColumnWidthModifications` are used to commit changes made to the width of the table or its columns when using the mouse drag gestures.

```
public void commitTableWidthModification(AuthorDocumentController authorDocumentController,
    int newTableWidth, String tableCellsTagName) throws AuthorOperationException {
    if ("td".equals(tableCellsTagName)) {
        if (newTableWidth > 0) {
            if (tableElement != null) {
                String newWidth = String.valueOf(newTableWidth);

                authorDocumentController.setAttribute(
                    "width",
                    new AttrValue(newWidth),
                    tableElement);
            } else {
                throw new AuthorOperationException("Cannot find the element representing the table.");
            }
        }
    }
}

public void commitColumnWidthModifications(AuthorDocumentController authorDocumentController,
    WidthRepresentation[] colWidths, String tableCellsTagName) throws AuthorOperationException {
    if ("td".equals(tableCellsTagName)) {
        if (colWidths != null && tableElement != null) {
            if (colsStartOffset >= 0 && colsEndOffset >= 0 && colsStartOffset < colsEndOffset) {
                authorDocumentController.delete(colsStartOffset,
                    colsEndOffset);

                String xmlFragment = createXMLFragment(colWidths);
                int offset = -1;
                AuthorElement[] header = tableElement.getElementsByLocalName("header");
                if (header != null && header.length > 0) {
                    // Insert the cols elements before the 'header' element
                    offset = header[0].getStartOffset();
                }
                if (offset == -1) {
                    throw new AuthorOperationException("No valid offset to insert the columns width specification.");
                }
                authorDocumentController.insertXMLFragment(xmlFragment, offset);
            }
        }
    }
}

private String createXMLFragment(WidthRepresentation[] widthRepresentations) {
    StringBuffer fragment = new StringBuffer();
    String ns = tableElement.getNamespace();
    for (int i = 0; i < widthRepresentations.length; i++) {
        WidthRepresentation width = widthRepresentations[i];
        fragment.append("<customcol");
        String strRepresentation = width.getWidthRepresentation();
        if (strRepresentation != null) {
            fragment.append(" width=\"" + width.getWidthRepresentation() + "\"");
        }
    }
}
```

```

    }
    if (ns != null && ns.length() > 0) {
        fragment.append(" xmlns=\"" + ns + "\\"");
    }
    fragment.append("/>");
}
return fragment.toString();
}

```

7. The following three methods are used to determine what type of column width specifications the table column width provider support. In our case all types of specifications are allowed:

```

public boolean isAcceptingFixedColumnWidths(String tableCellsTagName) {
    return true;
}

public boolean isAcceptingPercentageColumnWidths(String tableCellsTagName) {
    return true;
}

public boolean isAcceptingProportionalColumnWidths(String tableCellsTagName) {
    return true;
}

```



Note: The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK*, available as a Maven archetype [on the Oxygen XML Editor website](#).

In the listing below, the XML document contains the table element:

```





```

When no table column width provider is specified, the table has the following layout:

C1	C2	C3	C4
cs=1, rs=1	cs=1, rs=1	cs=1, rs=2	cs=1, rs=3
cs=1, rs=1	cs=1, rs=1		
cs=3, rs=1			

Figure 267: Table layout when no column width provider is specified

When the above implementation is configured, the table has the correct layout:

C1	C2	C3	C4
cs=1, rs=1	cs=1, rs=1	cs=1, rs=2	cs=1, rs=3
cs=1, rs=1	cs=1, rs=1		
cs=3, rs=1			

Figure 268: Columns with custom widths

Configuring a Table Cell Span Provider

In the sample documentation framework the `table` element can have cells that span over multiple columns and rows. As explained in [Configuring tables](#) on page 579, you need to indicate Oxygen XML Editor a method to determine the cell spanning. If you use the `cell` element attributes `rowspan` and `colspan` or `rows` and `cols`, Oxygen XML Editor can determine the cell spanning automatically. In our example the `td` element uses the attributes `row_span` and `column_span` that are not recognized by default. You will need to implement a Java extension class for defining the cell spanning.

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

1. Create the class `simple.documentation.framework.TableCellSpanProvider`. This class must implement the `ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider` interface.

```
import ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableCellSpanProvider
    implements AuthorTableCellSpanProvider {
```

2. The `init` method is taking as argument the `ro.sync.ecss.extensions.api.node.AuthorElement` that represents the XML `table` element. In our case the cell span is specified for each of the cells so you leave this

method empty. However there are cases like the table CALS model when the cell spanning is specified in the `table` element. In such cases you must collect the span information by analyzing the `table` element.

```
public void init(AuthorElement table) {
```

- The `getColSpan` method is taking as argument the table cell. The table layout engine will ask this `AuthorTableSpanSupport` implementation what is the column span and the row span for each XML element from the table that was marked as cell in the CSS using the property `display:table-cell`. The implementation is simple and just parses the value of `column_span` attribute. The method must return null for all the cells that do not change the span specification.

```
public Integer getColSpan(AuthorElement cell) {
    Integer colSpan = null;

    AttrValue attrValue = cell.getAttribute("column_span");
    if(attrValue != null) {
        // The attribute was found.
        String cs = attrValue.getValue();
        if(cs != null) {
            try {
                colSpan = new Integer(cs);
            } catch (NumberFormatException ex) {
                // The attribute value was not a number.
            }
        }
    }
    return colSpan;
}
```

- The row span is determined in a similar manner:

```
public Integer getRowSpan(AuthorElement cell) {
    Integer rowSpan = null;

    AttrValue attrValue = cell.getAttribute("row_span");
    if(attrValue != null) {
        // The attribute was found.
        String rs = attrValue.getValue();
        if(rs != null) {
            try {
                rowSpan = new Integer(rs);
            } catch (NumberFormatException ex) {
                // The attribute value was not a number.
            }
        }
    }
    return rowSpan;
}
```

- The method `hasColumnSpecifications` always returns `true` considering column specifications always available.

```
public boolean hasColumnSpecifications(AuthorElement tableElement) {
    return true;
}
```

 **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK*, available as a Maven archetype [on the Oxygen XML Editor website](#).

- In the listing below, the XML document contains the table element:

```
<table>
  <header>
    <td>C1</td>
    <td>C2</td>
    <td>C3</td>
    <td>C4</td>
  </header>
  <tr>
    <td>cs=1, rs=1</td>
    <td column_span="2" row_span="2">cs=2, rs=2</td>
    <td row_span="3">cs=1, rs=3</td>
  </tr>
```

```

<tr>
  <td>cs=1, rs=1</td>
</tr>
<tr>
  <td column_span="3">cs=3, rs=1</td>
</tr>
</table>

```

When no table cell span provider is specified, the table has the following layout:

The screenshot shows the Oxygen XML Editor interface with a document containing a table element. The table has four columns labeled C1, C2, C3, and C4. The first row contains a single cell with 'cs=1, rs=1'. The second row contains three cells with 'cs=2, rs=2' and one empty cell. The third row contains two cells with 'cs=1, rs=1' and one empty cell. The fourth row contains one cell with 'cs=3, rs=1' and one empty cell. The table is displayed with borders around each cell.

C1	C2	C3	C4
cs=1, rs=1	cs=2, rs=2	cs=1, rs=3	
cs=1, rs=1			
cs=3, rs=1			

Figure 269: Table layout when no cell span provider is specified

When the above implementation is configured, the table has the correct layout:

The screenshot shows the Oxygen XML Editor interface with the same document and table structure as Figure 269. However, the table now displays the correct layout according to the configuration. The first row has a single cell with 'cs=1, rs=1'. The second row has a single cell with 'cs=2, rs=2'. The third row has a single cell with 'cs=1, rs=1'. The fourth row has a single cell with 'cs=3, rs=1'. The table is displayed with borders around each cell.

C1	C2	C3	C4
cs=1, rs=1	cs=2, rs=2	cs=1, rs=3	
cs=1, rs=1			
cs=3, rs=1			

Figure 270: Cells spanning multiple rows and columns.

Configuring a Table Cell Row And Column Separator Provider

In the sample documentation framework the `table` element has separators between rows. As explained in [Configuring tables](#) on page 579 section which describes the CSS properties needed for defining a table, you need to indicate Oxygen

XML Editor a method to determine the way rows and columns are separated. If you use the `rowsep` and `colsep` cell element attributes, or your table is conforming to the CALS table model, Oxygen XML Editor can determine the cell separators. In the example there are no attributes defining the separators but we still want the rows to be separated. You will need to implement a Java extension.

-  **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

1. Create the class `simple.documentation.framework.TableCellSepProvider`. This class must implement the `ro.sync.ecss.extensions.api.AuthorTableCellSepProvider` interface.

```
import ro.sync.ecss.extensions.api.AuthorTableCellSepProvider;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableCellSepProvider implements AuthorTableCellSepProvider{
```

2. The `init` method is taking as argument the `ro.sync.ecss.extensions.api.node.AuthorElement` that represents the XML `table` element. In our case the separator information is implicit, it does not depend on the current table, so you leave this method empty. However there are cases like the table CALS model when the cell separators are specified in the `table` element - in that case you should initialize your provider based on the given argument.

```
public void init(AuthorElement table) {
```

3. The `getColSep` method is taking as argument the table cell. The table layout engine will ask this `AuthorTableCellSepProvider` implementation if there is a column separator for each XML element from the table that was marked as cell in the CSS using the property `display:table-cell`. In our case we choose to return `false` since we do not need column separators.

```
/**
 * @return false - No column separator at the right of the cell.
 */
@Override
public boolean getColSep(AuthorElement cellElement, int columnIndex) {
    return false;
}
```

4. The row separators are determined in a similar manner. This time the method returns `true`, forcing a separator between the rows.

```
/**
 * @return true - A row separator below each cell.
 */
@Override
public boolean getRowSep(AuthorElement cellElement, int columnIndex) {
    return true;
}
```

-  **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the [Oxygen SDK](#), available as a Maven archetype [on the Oxygen XML Editor website](#).

5. In the listing below, the XML document contains the `table` element:

```
<table>
  <header>
    <td>H1</td>
    <td>H2</td>
    <td>H3</td>
    <td>H4</td>
  </header>
  <tr>
    <td>C11</td>
    <td>C12</td>
    <td>C13</td>
    <td>C14</td>
  </tr>
  <tr>
    <td>C21</td>
```

```

<td>C22</td>
<td>C23</td>
<td>C24</td>
</tr>
<tr>
<td>C31</td>
<td>C32</td>
<td>C33</td>
<td>C34</td>
</tr>
</table>

```

When the borders for the `td` element are removed from the CSS, the row separators become visible:

H1	H2	H3	H4
C11	C12	C13	C14
C21	C22	C23	C24
C31	C32	C33	C34

Figure 271: Row separators provided by the Java implementation.

Configuring an Unique Attributes Recognizer

The `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` interface can be implemented if you want to provide for your framework the following features:

- **Automatic ID generation** - You can automatically generate unique IDs for newly inserted elements. Implementations are already available for the DITA and DocBook frameworks. The following methods can be implemented to accomplish this: `assignUniqueIDs(int startOffset, int endOffset)`, `isAutoIDGenerationActive()`
- **Avoiding copying unique attributes when "Split" is called inside an element** - You can split the current block element by pressing the "Enter" key and then choosing "Split". This is a very useful way to create new paragraphs, for example. All attributes are by default copied on the new element but if those attributes are IDs you sometimes want to avoid creating validation errors in the editor. Implementing the following method, you can decide whether an attribute should be copied or not during the split: `boolean copyAttributeOnSplit(String attrQName, AuthorElement element)`



Tip:

The `ro.sync.ecss.extensions.commons.id.DefaultUniqueAttributesRecognizer` class is an implementation of the interface which can be extended by your customization to provide easy assignation of IDs in your framework. You can also check out the DITA and DocBook implementations of `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` to see how they were implemented and connected to the extensions bundle.

Configuring an XML Node Renderer Customizer

You can use this API extension to customize the way an XML node is rendered in the **Author Outline** view, **Author** breadcrumb navigation bar, **Text mode Outline** view, content completion assistant window or **DITA Maps Manager** view.



Note: Oxygen XML Editor uses `XMLNodeRendererCustomizer` implementations for the following frameworks: DITA, DITAMap, DocBook 4, DocBook 5, TEI P4, TEI P5, XHTML, XSLT, and XML Schema.



Note: The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor website](#). Also it is available in the [Oxygen SDK Maven Project](#).

There are two methods to provide an implementation of `ro.sync.exml.workspace.api.node.customizer.XMLNodeRendererCustomizer`:

- as a part of a bundle - returning it from the `createXMLNodeCustomizer()` method of the [ExtensionsBundle](#) associated with your document type in the **Document type** dialog box, **Extensions** tab, **Extensions bundle** field.
- as an individual extension - associated with your document type in the **Document type** dialog box, **Extensions** tab, **Individual extensions** section, **XML node renderer customizer** field.

Support for Retina/HiDPI Displays

To support Retina or HiDPI displays, the icons provided by the `XMLNodeRendererCustomizer` should be backed up by a copy of larger size using the proper [Retina/HiDPI naming convention](#).

For example, for the `title` element, if the `XMLNodeRendererCustomizer` returns the path `${framework}/images/myImg.png`, then in order to support Retina images with a scaling factor of 2, an extra file (`myImg@2x.png`) should be added to the same images directory (`${framework}/images/myImg@2x.png`). If the higher resolution icon (the `@2x` file) does not exist, the *normal* icon is scaled and used instead.

For more information about using Retina/HiDPI images, refer to the [Using Retina/HiDPI Images in Author Mode](#) section.

Customizing the Default CSS of a Document Type

The easiest way to customize the default CSS stylesheet of a document type is to create a new CSS stylesheet, save it in the same folder as the default CSS, and set the new stylesheet as the default CSS for the document type.

For example, to customize the default CSS for DITA documents by changing the background color of the `task` and `topic` elements to red, follow the following steps:

1. First, create a new CSS stylesheet named `my_dita.css` and save it in the folder `[OXYGEN_DIR]/frameworks/dita/css_classed`, where the default stylesheet named `dita.css` is located. The new stylesheet `my_dita.css` contains:

```
@import "dita.css";
task, topic{
    background-color:red;
}
```

2. To set the new stylesheet as the default CSS stylesheet for DITA documents, [open the Preferences dialog box](#) and go to **Document Type Association**. Select the DITA document type and press the **Edit** button. In the **Author** tab, change the URI of the default CSS stylesheet from `${framework}/css_classed/dita.css` to `${framework}/css_classed/my_dita.css`.

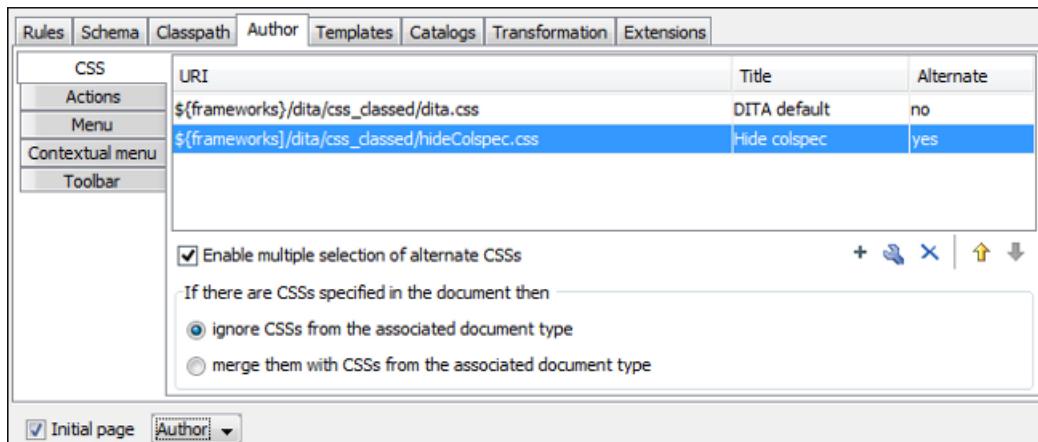


Figure 272: Set the location of the default CSS stylesheet

3. Press **OK** in all the dialog boxes to validate the changes. You can now edit DITA documents based on the new CSS stylesheet. You can also edit the new CSS stylesheet itself and see its effects on rendering DITA XML documents in the **Author** mode by running the *Refresh* action that is available on the **Author** toolbar and in the **DITA** menu.

Document Type Sharing

Oxygen XML Editor allows you to share the customizations for a specific XML type by creating your own *Document Type* in the **Document Type Association** preferences page.

A document type can be shared between authors as follows:

- Save it externally in a separate framework folder in the `[OXYGEN_DIR]/frameworks` directory.
-  **Important:** For this approach to work, have the application installed to a folder with full write access.

Please follow these steps:

1. Go to `[OXYGEN_DIR]/frameworks` and create a directory for your new framework (name it for example `custom_framework`). This directory will contain resources for your framework (CSS files, new file templates, schemas used for validation, catalogs). See the **DocBook** framework structure from the `[OXYGEN_DIR]/frameworks/docbook` as an example.
2. Create your custom document type and save it externally, in the `custom_framework` directory.
3. Configure the custom document type according to your needs, take special care to make all file references relative to the `[OXYGEN_DIR]/frameworks` directory by using the `#{frameworks}` editor variable. The *Author Developer Guide* contains all details necessary for creating and configuring a new document type.
4. If everything went fine then you should have a new configuration file saved in: `[OXYGEN_DIR]/frameworks/custom_framework/custom.framework` after the Preferences are saved.
5. Then, to share the new framework directory with other users, have them copy it to their `[OXYGEN_DIR]/frameworks` directory. The new document type will be available in the list of Document Types when Oxygen XML Editor starts.



Note: In case you have a `frameworks` directory stored on your local drive, you can also go to the **Document Type Association > Locations** preferences page and add your `frameworks` directory in the **Additional frameworks directories** list.

- Save the document type at project level in the **Document Type Association** preferences page.

Please see the following steps:

1. On your local drive, create a directory with full write access, containing the Oxygen XML Editor project file and associated document type resources (CSS files, new file templates, schemas used for validation, catalogs).
 2. Start Oxygen XML Editor, go to the *Project view* and create a project. Save it in the newly created directory.
 3. In the **Document Type Association** preferences page, select **Project Options** at the bottom of the page.
 4. Create your custom document type using the default **internal** storage for it. It will actually be saved in the previously chosen Oxygen XML Editor project `.xpr` file.
 5. Configure the custom document type according to your needs, take special care to make all file references relative to the project directory by using the `#{pd}` editor variable. The *Author Developer Guide* contains all details necessary for creating and configuring a new document type.
 6. You can then share the new project directory with other users. When they open the customized project file in the *Project view*, the new document type becomes available in the list of Document Types.
- Deploy your document type configuration *as an add-on*.

Adding Custom Persistent Highlights

The Author API allows you to create or remove custom persistent highlights, set their properties, and customize their appearance. They get serialized in the XML document as processing instructions, having the following format:

```
<?oxy_custom_start prop1="val1"....?> xml content <?oxy_custom_end?>
```

The functionality is available in the `AuthorPersistentHighlighter` class, accessible through `AuthorEditorAccess#getPersistentHighlighter()` method. For more information, see JavaDoc online at: <http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/index.html>

Providing Additional Documentation for XML Elements and Attributes

Oxygen XML Editor gathers documentation from the associated schemas (DTDs, XML Schema, RelaxNG) and presents it for each element or attribute. For example, if you open the **Content Completion Assistant** for a recognized XML vocabulary, documentation is displayed for each element provided by the associated schema. Similar information is displayed when you hover over tag names presented in the **Elements** view. If you hover over attributes in the **Attributes** view you also see information about each attribute, gathered from the same schema.

If you have a document type configuration set up for your XML vocabulary, there is a special XML configuration file that can be added to provide additional documentation information or links to specification web pages for certain elements and attributes. To provide this additional information, follow these steps:

1. Create a new folder in the configuration directory for the document type. For instance:
OXYGEN_INSTALL_DIR/frameworks/dita/styleguide.
2. Use the **New** document wizard to create a file using the `Oxygen content completion styleguide` file template.
3. Make the appropriate changes to your custom mapping file. For example, you can look at how the DITA mapping file is configured:
OXYGEN_INSTALL_DIR/frameworks/dita/styleguide/contentCompletionElementsMap.xml. The associated XML Schema contains additional details about how each element and attribute is used in the mapping file.
4. Save the file in the folder created in step 1, using the fixed name: `contentCompletionElementsMap.xml`.
5. *Open the **Preferences** dialog box*, go to **Document Type Association**, and edit the document type configuration for your XML vocabulary. Now you need to indicate where Oxygen XML Editor will locate your mapping file by doing one of the following:
 - In the **Classpath** tab add a link to the newly created folder.
 - In the **Catalogs** tab *add a new catalog file*. The selected file needs to contain the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
"http://www.casis-open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn: oasis:names:tc:entity:xml:catalog">
  <uri name="http://www.oxygenxml.com/{processed_dt_name}/styleguide/contentCompletionElementsMap.xml"
uri="contentCompletionElementsMap.xml"/>
</catalog>
```

where `{processed_dt_name}` is the name of the document type in lower case and with spaces replaced by underscores.



Note: If Oxygen XML Editor finds a mapping file in both locations, the one in the **Catalogs** tab takes precedence.

6. Re-open the application and open an XML document.

In the **Content Completion Assistant** you should see the additional annotations for each element.

Configuring the Proposals in the Content Completion Assistant

Oxygen XML Editor gathers information from the associated schemas (DTDs, XML Schema, RelaxNG) to determine the proposals that appear in the **Content Completion Assistant**. Oxygen XML Editor also includes support that allows you to configure the possible attribute or element values for the proposals. To do so, a configuration file can be used, along with the associated schema, to add or replace possible values for attributes or elements that are proposed in the **Content Completion Assistant**. An example of a specific use-case is if you want the **Content Completion Assistant** to propose several possible values for the language code whenever you use an `xml:lang` attribute.

To configure content completion proposals, follow these steps:

1. Create a new resources folder (if it does not already exist) in the frameworks directory for the document type. For instance: OXYGEN_INSTALL_DIR/frameworks/dita/resources.
2. *Open the Preferences dialog box* and go to **Document Type Association**. Edit the document type configuration for your XML vocabulary, and in the **Classpath** tab add a link to that resources folder.
3. Use the **New** document wizard to create a configuration file using the **Content Completion Configuration** file template.
4. Make the appropriate changes to your custom configuration file. The file template includes details about how each element and attribute is used in the configuration file.
5. Save the file in the resources folder, using the fixed name: cc_value_config.xml.
6. Re-open the application and open an XML document. In the **Content Completion Assistant** you should see your customizations.

The Configuration File

The configuration file is composed of a series of `match` instructions that will match either an element or an attribute name. A new value is specified inside one or more `item` elements, which are grouped inside an `items` element. The behavior of the `items` element is specified with the help of the `action` attribute, which can have any of the following values:

- `append` - Adds new values to appear in the proposals list (default value).
- `addIfEmpty` - Adds new values to the proposals list, only if no other values are contributed by the schema.
- `replace` - Replaces the values contributed by the schema with new values to appear in the proposals list.

The values in the configuration file can be specified either directly or by calling an external XSLT file that will extract data from any external source.

Example - Specifying Values Directly

```
<!-- Replaces the values for an element with the local name "lg", from the given namespace -->
<match elementName="lg" elementNS="http://www.oxygenxml.com/ns/samples">
  <items action="replace">
    <item value="stanza"/>
    <item value="refrain"/>
  </items>
</match>

<!-- Adds two values for an attribute with the local name "type", from any namespace -->
<match attributeName="type">
  <items>
    <item value="stanza"/>
    <item value="refrain"/>
  </items>
</match>
```

Example - Calling an External XSLT Script

```
<xslt href="../xsl/get_values_from_db.xsl" useCache="false" action="replace"/>
```

In this example, the `get_values_from_db.xsl` is executed in order to extract values from a database.



Note: A comprehensive XSLT sample is included in the **Content Completion Configuration** file template.

CSS Support in Author

Author editing mode supports most CSS 2.1 selectors, numerous CSS 2.1 properties, and some CSS 3 selectors. Oxygen XML Editor also supports stylesheets coded with the LESS dynamic stylesheet language. Also, some custom functions and properties that extend the W3C CSS specification, and are useful for URL and string manipulation, are available to developers who create **Author** editing frameworks.

Handling CSS Imports

When a CSS document contains imports to other CSS documents, the references are also passed through the XML catalog URI mappings in order to determine an indirect CSS referenced location.

You can have a CSS import like:

```
@import "http://host/path/to/location/custom.css";
```

and then add your own XML catalog file that maps the location to a custom CSS in the *XML / XML Catalog* preferences page:

```
<uri name="http://host/path/to/location/custom.css" uri="path/to/custom.css"/>
```

In addition, you can add the following mapping in your XML Catalog file:

```
<uri name="http://www.oxygenxml.com/extensions/author/css/userCustom.css" uri="path/to/custom.css"/>
```

This extra mapped CSS location will be parsed every time the application processes the CSS stylesheets used to render the opened XML document in the visual **Author** editing mode. This allows your custom CSS to be used without the need to modify all other CSS stylesheets contributed in the document type configuration.

Selecting and Combining Multiple CSS Styles

Oxygen XML Editor provides a **Styles** drop-down list on the **Author Styles** toolbar that allows you to select one main (*non-alternate*) CSS style and multiple *alternate* CSS styles. An option in the preferences can be enabled to allow the alternate styles to behave like layers and be combined with the main CSS style. This makes it easy to change the look of the document.

An example of a common use case is when content authors want to use custom styling within a document. You can select a main CSS stylesheet that styles the whole document and then apply alternate styles, as layers, to specific parts of the document. In the subsequent figure, a DITA document has the **Century** style selected for the main CSS and the alternate styles **Full width**, **Show table column specification**, **Hints**, and **Inline actions** are combined for additive styling to specific parts of the document.

-  **Note:** Oxygen XML Editor comes with a set of predefined CSS layer stylesheets for DITA documents only, but the support is available for all other document types.
-  **Tip:** The **Hints** style displays tooltips throughout DITA documents that offer additional information to help you with the DITA structure. The **Inline actions** style displays possible elements that are allowed to be inserted at various locations throughout DITA documents.

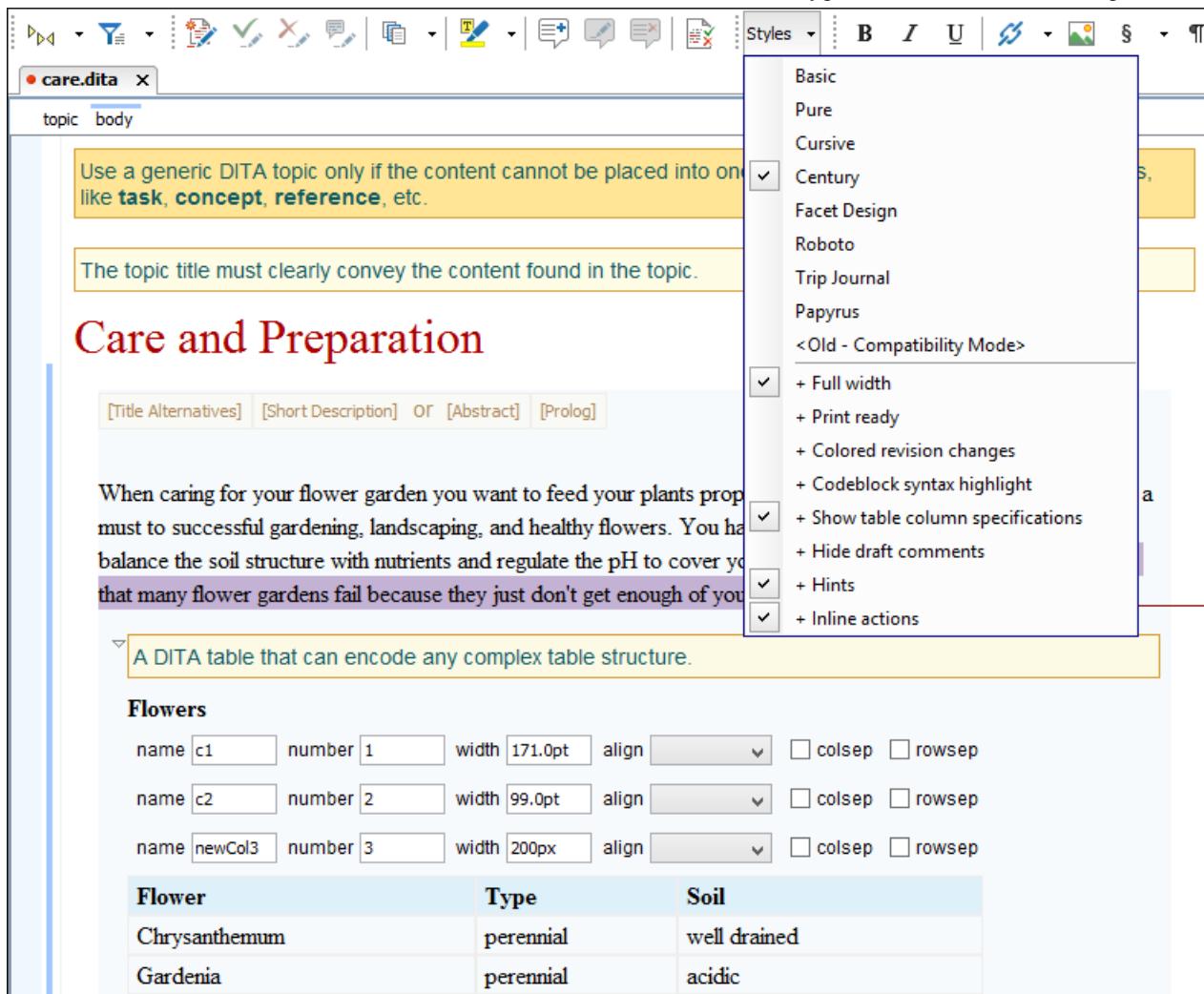


Figure 273: Styles Drop-down List in a DITA Document

The main and alternate styles that are listed in the **Styles** drop-down list can be controlled in the [Document Type Association dialog box](#). To access it, follow these steps:

1. [Open the Preferences dialog box](#).
2. Go to **Document Type Association**.
3. Select the appropriate document type and press the **Edit** button.

The CSS styles associated with the particular document type are listed in the **Author** tab.

The names listed in the **Styles** drop-down list match the values in the **Title** column. The value in the **Alternate** column determines whether it is a main or alternate CSS. If the value is *no* it is a main CSS. If the value is *yes* it is an alternate CSS and the style can be combined with a main CSS or other alternate styles when using the **Styles** drop-down list.

 **Note:** To group alternate styles into categories, use a vertical bar character (|) in the **Title** column. The part before the vertical bar will be rendered as a menu entry in the **Styles** dropdown, while the part after the vertical bar will be rendered as the style's name.

Example: Let's suppose that we add two alternate stylesheets, with the **Title** column set to **User Assistance|Hints** and **User Assistance|Inline Actions**. Oxygen XML Editor will add in the **Styles** dropdown a **User Assistance** submenu, containing the **Hints** and **Inline Actions** items.

A developer can add, edit, or delete styles from this dialog box to control the main and alternate styles associated to the particular document type. Notice that the CSS styles shown in the following figure match the styles listed in the drop-down list in the previous figure.

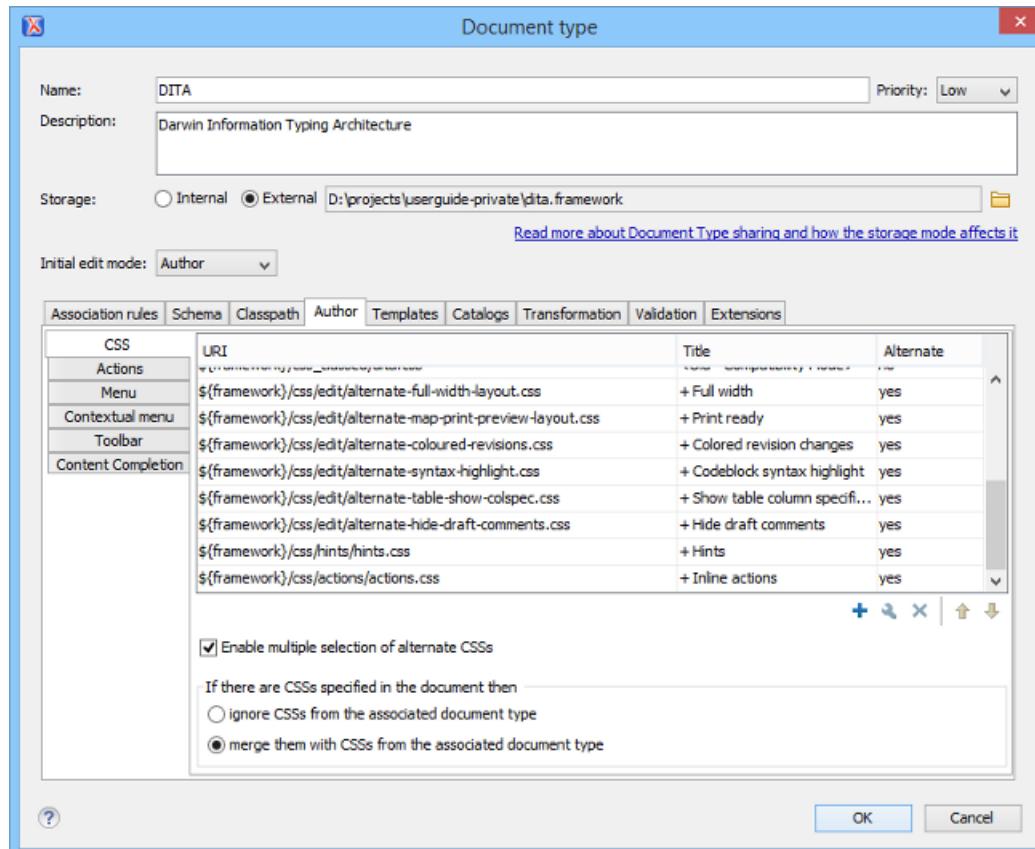


Figure 274: Main and Alternate CSS Styles in the Document Type Association Dialog Box

The **Enable multiple selection of alternate CSSs** box at the bottom of the pane must be checked in order for the alternate styles to be combined. If this option is disabled, the alternate styles are treated like main CSS styles and you can only select one at a time. By default, this option is enabled for DITA documents. There are also a few radio button options to specify how to handle the CSS if there are CSS styles specified in the document. You can choose to *ignore* or *merge* them.

The selections from the **Styles** drop-down list are persistent, meaning that Oxygen XML Editor will remember the selections when subsequent documents are opened.



Note: The application also supports working directly with LESS stylesheets, instead of CSS.

The oxygen Media Type

The CSS stylesheets can specify how a document is presented on different types of media (on the screen, paper, etc.). You can specify that some of the selectors from your CSS should be taken into account only in the Oxygen XML Editor **Author** mode and ignored in other media types. This can be accomplished by using the oxygen media type.

```
b{
    font-weight:bold;
    display:inline;
}

@media oxygen{
    b{
        text-decoration:underline;
    }
}
```

```
}
```

This example results in the text being bold if the document is opened in a web browser that does not recognize @media oxygen, while the text is bold and underlined when opened in Oxygen XML Editor **Author** mode.

You can also use the oxygen media type to specify CSS selectors to be applied in certain operating systems or platforms by using the `os` and `platform` properties. For example, you can specify a set of style rules for displaying Oxygen XML Editor in Windows, and a different set of style rules for Mac OS. The supported properties are as follows:

- **os** - The possible values are: `win`, `linux`, or `mac`.
- **platform** - The possible values are: `standalone` and `eclipse`

```
@media oxygen AND (os:"win") AND (platform:"standalone")
  p{
    content:"PPP";
  }
}
```

Standard W3C CSS Supported Features

Oxygen XML Editor supports most of the CSS Level 3 selectors and most of the CSS Level 2.1 properties

Supported CSS Selectors

Expression	Name	CSS Level	Description / Example
<code>*</code>	Universal selector	CSS Level 2	Matches any element
<code>E</code>	Type selector	CSS Level 2	Matches any E element (i. e. an element with the local name E)
<code>E F</code>	Descendant selector	CSS Level 2	Matches any F element that is a descendant of an E element.
<code>E > F</code>	Child selectors	CSS Level 2	Matches any F element that is a child of an element E.
<code>E:lang(c)</code>	Language pseudo-class	CSS Level 2	Matches element of type E if it is in (human) language c (the document language specifies how language is determined).
<code>E + F</code>	Adjacent selector	CSS Level 2	Matches any F element immediately preceded by a sibling element E.
<code>E ~ F</code>	General sibling selector	CSS Level 3	Matches any F element preceded by a sibling element E.
<code>E[foo]</code>	Attribute selector	CSS Level 2	Matches any E element with the "foo" attribute set (whatever the value).
<code>E[foo="warning"]</code>	Attribute selector with value	CSS Level 2	Matches any E element whose "foo" attribute value is exactly equal to "warning".
<code>E[foo~= "warning"]</code>	Attribute selector containing value	CSS Level 2	Matches any E element whose "foo" attribute value is a list of space-separated values, one of which is exactly equal to "warning".

Expression	Name	CSS Level	Description / Example
E[lang = "en"]	Attribute selector containing hyphen separated values	CSS Level 2	Matches any E element whose "lang" attribute has a hyphen-separated list of values beginning (from the left) with "en".
E:before and E:after	Pseudo elements	CSS Level 2	The ':before' and ':after' pseudo-elements can be used to insert generated content before or after an element's content.
E:before(n) and E:after(n)	Pseudo elements	CSS Level 3	Multiple ':before(n)' and ':after(n)' pseudo-elements can be used to insert content before or after the content of an element (or other pseudo-element). For more information, see the W3C CSS3 pseudo elements site .
E:first-child	The first-child pseudo-class	CSS Level 2	Matches element E when E is the first child of its parent.
E:not(s)	Negation pseudo-class	CSS Level 2	An E element that does not match simple selector s.
E:hover	The hover pseudo-class	CSS Level 2	The :hover pseudo-class applies while the user designates an element with a pointing device, but does not necessarily activate it. When moving the pointing device over an element, all the parent elements up to the root are taken into account.
E:focus	The focus pseudo-class	CSS Level 2	The :focus pseudo-class applies while an element has the focus (accepts keyboard input).
E#myid	The ID selector	CSS Level 2	Matches any E element with ID equal to "myid". ! Important: Limitation: In Oxygen XML Editor the match is performed taking into account only the attributes with the exact name: "id".
E[att^="val"]	Substring matching attribute selector	CSS Level 3	An E element whose att attribute value begins exactly with the string val.
E[att\$="val"]	Substring matching attribute selector	CSS Level 3	An E element whose att attribute value ends exactly with the string val.
E[att*="val"]	Substring matching attribute selector	CSS Level 3	An E element whose att attribute value contains the substring val.
E:root	Root pseudo-class	CSS Level 3	Matches the root element of the document. In HTML, the root element is always the HTML element.

Expression	Name	CSS Level	Description / Example
E:empty	Empty pseudo-class	CSS Level 3	An E element which has no text or child elements.
E:nth-child(n)	The nth-child pseudo-class	CSS Level 3	An E element, the nth child of its parent.
E:nth-last-child(n)	The nth-last-child pseudo-class	CSS Level 3	An E element, the nth child of its parent, counting from the last one.
E:nth-of-type(n)	The nth-of-type pseudo-class	CSS Level 3	An E element, the nth sibling of its type.
E:nth-last-of-type(n)	The nth-last-of-type pseudo-class	CSS Level 3	An E element, the nth sibling of its type, counting from the last one.
E:last-child	The last-child pseudo-class	CSS Level 3	An E element, last child of its parent.
E:first-of-type	The first-of-type pseudo-class	CSS Level 3	An E element, first sibling of its type.
E:last-of-type	The last-of-type pseudo-class	CSS Level 3	An E element, last sibling of its type.
E:only-child	The only-child pseudo-class	CSS Level 3	An E element, only child of its parent.
E:only-of-type	The only-of-type pseudo-class	CSS Level 3	An E element, only sibling of its type.
ns E	Element namespace selector	CSS Level 3	An element that has the local name E and the namespace given by the prefix ns. The namespace prefix can be bound to an URI by the at-rule: @namespace ns "http://some_namespace_uri"; See Namespace Selector on page 597.
E !>F	The subject selector	CSS Level 4 (experimental)	An element that has the local name E and has a child F. See Subject Selector on page 599.

Namespace Selector

In the CSS 2.1 standard the element selectors are ignoring the namespaces of the elements they are matching. Only the local name of the elements are considered in the selector matching process.

Oxygen XML Editor Author uses a different approach similar to the CSS Level 3 specification. If the element name from the CSS selector is not preceded by a namespace prefix it is considered to match an element with the same local name as the selector value and ANY namespace, otherwise the element must match both the local name and the namespace.

In CSS up to version 2.1 the name tokens from selectors are matching all elements from ANY namespace that have the same local name. Example:

```
<x:b xmlns:x="ns_x"/>
<y:b xmlns:y="ns_y"/>
```

Are both matched by the rule:

```
b {font-weight:bold}
```

Starting with CSS Level 3 you can create selectors that are namespace aware.

Defining both prefixed namespaces and the default namespace

Given the namespace declarations:

```
@namespace sync "http://sync.example.org";
@namespace "http://example.com/foo";
```

In a context where the default namespace applies:

sync|A

represents the name A in the `http://sync.example.org` namespace.

|B

represents the name B that belongs to NO NAMESPACE.

***|C**

represents the name C in ANY namespace, including NO NAMESPACE.

D

represents the name D in the `http://example.com/foo` namespace.

Defining only prefixed namespaces

Given the namespace declaration:

```
@namespace sync "http://sync.example.org";
```

Then:

sync|A

represents the name A in the `http://sync.example.org` namespace.

|B

represents the name B that belongs to NO NAMESPACE.

***|C**

represents the name C in ANY namespace, including NO NAMESPACE.

D

represents the name D in ANY namespace, including NO NAMESPACE.

Defining prefixed namespaces combined with pseudo-elements

To match the `def` element its namespace will be declared, bind it to the `abs` prefix, and then write a CSS rule:

```
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";
```

Then:

abs|def

represents the name "def" in the
`http://www.oxygenxml.com/sample/documentation/abstracts` namespace.

abs|def:before

represents the :before pseudo-element of the "def" element from the
<http://www.oxygenxml.com/sample/documentation/abstracts> namespace.

Subject Selector

Oxygen XML Editor Author supports the subject selector described in CSS Level 4 (currently a working draft at W3C <http://www.w3.org/TR/selectors4/>). This selector matches a structure of the document, but unlike a compound selector, the styling properties are applied to the subject element (the one marked with "!"") instead of the last element from the path.

The subject of the selector can be explicitly identified by appending an exclamation mark (!) to one of the compound selectors in a selector. Although the element structure that the selector represents is the same with or without the exclamation mark, indicating the subject in this way can change which compound selector represents the subject in that structure.

```
table! > caption {
    border: 1px solid red;
}
```

A border will be drawn to the table elements that contain a caption as direct child.

This is different from:

```
table > caption {
    border: 1px solid red;
}
```

which draws a border around the caption.



Important: As a limitation of the current implementation the general descendant selectors are taken into account as direct child selectors. For example the two CSS selectors are considered equivalent:

```
a! b c
```

and:

```
a! > b > c
```

Supported CSS Properties

Oxygen XML Editor validates all CSS 2.1 properties, but does not render *aural* and *paged* categories properties in **Author** mode, as well as some of the values of the *visual* category that are listed below under the **Ignored Values** column. For the Oxygen XML Editor-specific (extension) CSS properties, go to [Oxygen XML Editor CSS Extensions](#) on page 607.

Name	Rendered Values	Ignored Values
'background-attachment'		ALL
'background-color'	<color> inherit	transparent
'background-image'	<uri> none inherit	
'background-position'	top right bottom left center	<percentage> <length>
'background-repeat'	repeat repeat-x repeat-y no-repeat inherit	

Name	Rendered Values	Ignored Values
'background'		ALL
'border-collapse'		ALL
'border-color'	<color> inherit	transparent
'border-spacing'		ALL
'border-style'	<border-style> inherit	
'border-top' 'border-right' 'border-bottom' 'border-left'	[<border-width> <border-style> 'border-top-color'] inherit	
'border-top-color' 'border-right-color' 'border-bottom-color' 'border-left-color'	<color> inherit	transparent
'border-top-style' 'border-right-style' 'border-bottom-style' 'border-left-style'	<border-style> inherit	
'border-top-width' 'border-right-width' 'border-bottom-width' 'border-left-width'	<border-width> inherit	
'border-width'	<border-width> inherit	
'border'	[<border-width> <border-style> 'border-top-color'] inherit	
'bottom'		ALL
'caption-side'		ALL
'clear'		ALL
'clip'		ALL
'color'	<color> inherit	
'content'	normal none [<string> <URI> <counter> attr(<identifier>) open-quote close-quote]+ inherit	no-open-quote no-close-quote
'counter-increment'	[<identifier> <integer> ?]+ none inherit	
'counter-reset'	[<identifier> <integer> ?]+ none inherit	
'cursor'		ALL
'direction'	ltr rtl inherit	

Name	Rendered Values	Ignored Values
'display'	inline block list-item table table-row-group table-header-group table-footer-group table-row table-column-group table-column table-cell table-caption none inherit	run-in inline-block inline-table - considered block
'empty-cells'	show hide inherit	
'float'		ALL
'font-family'	[[<family-name> <generic-family>] [, <family-name> <generic-family>]*] inherit	
'font-size'	<absolute-size> <relative-size> <length> <percentage> inherit	
'font-style'	normal italic oblique inherit	
'font-variant'		ALL
'font-weight'	normal bold bolder lighter 100 200 300 400 500 600 700 800 900 inherit	
'font'	[['font-style' 'font-weight']? 'font-size' [/ 'line-height']? 'font-family'] inherit	'font-variant' 'line-height' caption icon menu message-box small-caption status-bar
'height'		ALL
'left'		ALL
'letter-spacing'	normal <length> inherit	
'line-height'	normal <number> <length> <percentage> inherit	
'list-style-image'		ALL
'list-style-position'		ALL
'list-style-type'	disc circle square decimal lower-roman upper-roman lower-latin upper-latin lower-alpha upper-alpha -oxy-lower-cyrillic-ru	lower-greek armenian georgian

Name	Rendered Values	Ignored Values
	-oxy-lower-cyrillic-uk -oxy-upper-cyrillic-ru -oxy-upper-cyrillic-uk box diamond check hyphen none inherit	
'list-style'	['list-style-type'] inherit	'list-style-position' 'list-style-image'
'margin-right' 'margin-left'	<margin-width> inherit auto	
'margin-top' 'margin-bottom'	<margin-width> inherit	
'margin'	<margin-width> inherit auto	
'max-height'		ALL
'max-width'	<length> <percentage> none inherit - supported for inline, block-level, and replaced elements, e.g. images, tables, table cells.	
'min-height'	Absolute values, such as 230px, 1in, 7pt, 12em.	Values proportional to the parent element height, such as 30%.
'min-width'	<length> <percentage> inherit - supported for inline, block-level, and replaced elements, e. g. images, tables, table cells.	
'outline-color'		ALL
'outline-style'		ALL
'outline-width'		ALL
'outline'		ALL
'overflow'		ALL
'padding-top' 'padding-right' 'padding-bottom' 'padding-left'	<padding-width> inherit	
'padding'	<padding-width> inherit	
'position'	absolute fixed - supported for block display elements, relative - supported for block and inline display elements	absolute fixed not supported for inline display elements
'quotes'		ALL
'right'		ALL
'table-layout'	auto	fixed inherit
'text-align'	left right center inherit	justify

Name	Rendered Values	Ignored Values
'text-decoration'	none [underline overline line-through] inherit	blink
'text-decoration-style'	solid double dotted dashed wavy initial inherit	
'text-indent'		ALL
'text-transform'	none capitalize uppercase lowercase inherit	
'top'		ALL
'unicode-bidi'	bidi-override normal embed inherit	
'vertical-align'	baseline sub super top text-top middle bottom text-bottom inherit	<percentage> <length>
'visibility'	visible hidden inherit -oxy-collapse-text	collapse
'white-space'	normal pre nowrap pre-wrap pre-line	
'width'	<length> <percentage> auto inherit - supported for inline, block-level, and replaced elements, e.g. images, tables, table cells.	
'word-spacing'		ALL
'z-index'		ALL

Transparent Colors

CSS3 supports RGBA colors. The RGBA declaration allows you to set opacity (via the Alpha channel) as part of the color value. A value of 0 corresponds to a completely transparent color, while a value of 1 corresponds to a completely opaque color. To specify a value, you can use either a *real* number between 0 and 1, or a percent.

RGBA color

```
personnel:before {
    display:block;
    padding: 1em;
    font-size: 1.8em;
    content: "Employees";
    font-weight: bold;
    color:#EEEEEE;
    background-color: rgba(50, 50, 50, 0.6);
}
```

The attr() Function: Properties Values Collected from the Edited Document.

In CSS Level 2.1 you may collect attribute values and use them as content *only* for the pseudo-elements. For instance the :before pseudo-element can be used to insert some content before an element. This is valid in CSS 2.1:

```
title:before{
    content: "Title id=" attr(id) ")";
}
```

If the title element from the XML document is:

```
<title id="title12">My title.</title>
```

Then the title will be displayed as:

```
title id=(title12) My title.
```

In Oxygen XML Editor Author the use of attr() function is available not only for the content property, but also for any other property. This is similar to the CSS Level 3 working draft:

<http://www.w3.org/TR/2006/WD-css3-values-20060919/#functional>. The arguments of the function are:

attr(attribute_name , attribute_type , default_value)

attribute_name

The attribute name. This argument is required.

attribute_type

The attribute type. This argument is optional. If it is missing, argument's type is considered string. This argument indicates what is the meaning of the attribute value and helps to perform conversions of this value. Oxygen XML Editor Author accepts one of the following types:

color

The value represents a color. The attribute may specify a color in different formats. Oxygen XML Editor Author supports colors specified either by name: red, blue, green, etc. or as an RGB hexadecimal value #FFEEFF.

url

The value is an URL pointing to a media object. Oxygen XML Editor Author supports only images. The attribute value can be a complete URL, or a relative one to the XML document. Please note that this URL is also resolved through the catalog resolver.

integer

The value must be interpreted as an integer.

number

The value must be interpreted as a float number.

length

The value must be interpreted as an integer.

percentage

The value must be interpreted relative to another value (length, size) expressed in percents.

em

The value must be interpreted as a size. 1 em is equal to the *font-size* of the relevant font.

ex

The value must be interpreted as a size. 1 ex is equal to the *height* of the x character of the relevant font.

px

The value must be interpreted as a size expressed in pixels relative to the viewing device.

mm

The value must be interpreted as a size expressed in millimeters.

cm

The value must be interpreted as a size expressed in centimeters.

in

The value must be interpreted as a size expressed in inches. 1 inch is equal to 2.54 centimeters.

pt

The value must be interpreted as a size expressed in points. The points used by CSS2 are equal to 1/72th of an inch.

pc

The value must be interpreted as a size expressed in picas. 1 pica is equal to 12 points.

default_value

This argument specifies a value that is used by default if the attribute value is missing. This argument is optional.

Usage samples for the attr() function

Consider the following XML instance:

```
<sample>
  <para bg_color="#AAAAFF">Blue paragraph.</para>
  <para bg_color="red">Red paragraph.</para>
  <para bg_color="red" font_size="2">Red paragraph with large font.</para>
  <para bg_color="#00AA00" font_size="0.8" space="4">
    Green paragraph with small font and margin.</para>
</sample>
```

The para elements have bg_color attributes with RGB color values like #AAAAFF. You can use the attr() function to change the elements appearance in the editor based on the value of this attribute:

```
background-color:attr(bg_color, color);
```

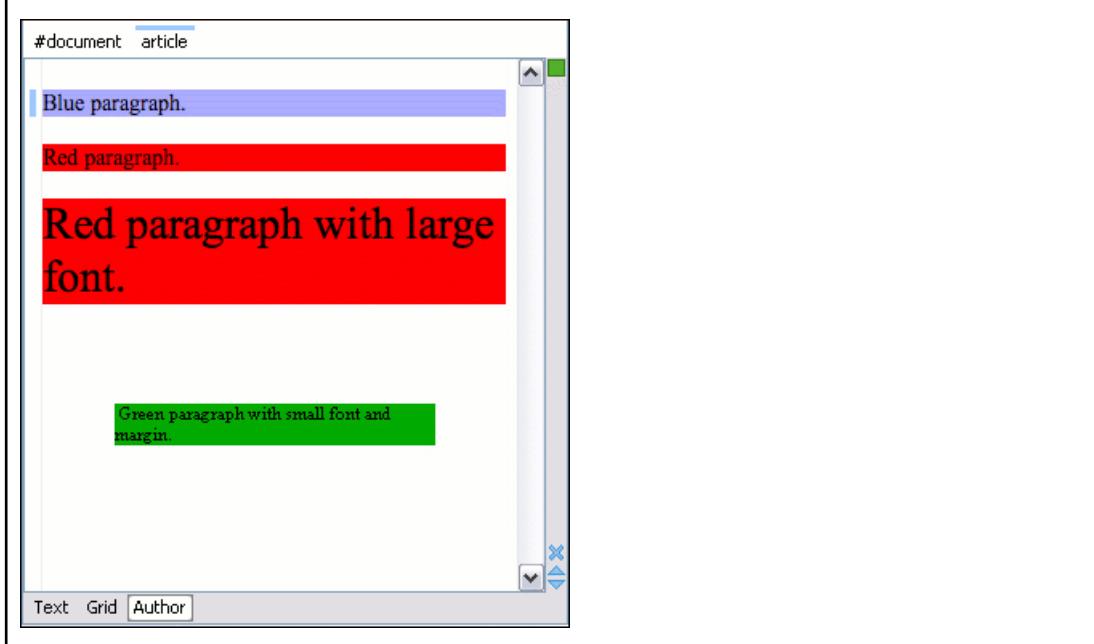
The attribute font_size represents the font size in *em* units. You can use this value to change the style of the element:

```
font-size:attr(font_size, em);
```

The complete CSS rule is:

```
para{
  display:block;
  background-color:attr(bg_color, color);
  font-size:attr(font_size, em);
  margin:attr(space, em);
}
```

The document is rendered as:



Supported CSS At-rules

Oxygen XML Editor supports some of the at-rules specified by CSS Level 2.1 and 3.

The @font-face at-rule

Oxygen XML Editor allows you to use custom fonts in the **Author** mode by specifying them in the CSS using the @font-face media type. Only the `src` and `font-family` CSS properties can be used for this media type.

```
@font-face{
    font-family:"Baroque Script";
    /*The location of the loaded TTF font must be relative to the CSS*/
    src:url("BaroqueScript.ttf");
}
```

The @media Rule

The @media rule allows you to set different style rules for various types of media in the same stylesheet. For example, you can set the font size to be different on the screen than on paper. Oxygen XML Editor supports several media types, allowing you to set the style rules for presenting a document on various media (on the screen, paper, etc.)

Supported Media Types

- `screen` - The styles marked with this media type are used only for rendering a document on the screen.
- `print` - The styles marked with this media type are used only for printing a document.
- `all` - The styles marked with this media type are used for rendering a document in all supported types of media.
- `oxygen` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor **Author** mode. For more information, see [The oxygen Media Type](#) on page 594 section.
- `oxygen-high-contrast-black` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor **Author** mode, on a Windows High Contrast Theme with a black background.
- `oxygen-high-contrast-white` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor **Author** mode, on a Windows High Contrast Theme with a white background.

```
@media oxygen{
b{
    text-decoration:underline;
}
}@media oxygen-high-contrast-white{
b{
```

```
    font-weight:bold;
}
```

Supported Properties

Oxygen XML Editor also supports a few properties to set specific style rules that depend upon the size of the visible area in **Author** mode. These supported properties are as follows:

- **min-width** - The styles selected in this property are applied if the visible area in **Author** mode is equal to or greater than the specified value.
- **max-width** - The styles selected in this property are applied if the visible area in **Author** mode is less than or equal to the specified value.

```
@media (min-width:500px){
  p{
    content:'xxx';
  }
}
@media (max-width:700px){
  p:after{
    content:'yyy';
  }
}
```

Oxygen XML Editor CSS Extensions

CSS stylesheets provide support for displaying documents. When editing non-standard documents, Oxygen XML Editor CSS extensions are useful.

Examples of how they can be used:

- Property for marking foldable elements in large files.
- Enforcing a display mode for the XML tags, regardless of the current mode selected by the user.
- Constructing a URL from a relative path location.
- String processing functions.

Additional CSS Selectors

Oxygen XML Editor Author provides support for selecting additional types of nodes. These custom selectors apply to: *document*, *doctype sections*, *processing-instructions*, *comments*, *CDATA sections*, *reference sections*, and *entities*.

Processing-instructions are not displayed by default. To display them, [open the Preferences dialog box](#), go to **Editor > Author**, and select **Show processing instructions**.



Note: The custom selectors are presented in the default CSS for **Author** mode and all of their properties are marked with an */important* flag. For this reason, you have to set the */important* flag on each property of the custom selectors from your CSS to be applicable.

For the custom selectors to work in your CSS stylesheets, declare the Author extensions namespace at the beginning of the stylesheet documents:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
```

- The *oxy/document* selector matches the entire document:

```
oxy|document {
  display:block !important;
}
```

- The following example changes the rendering of doctype sections:

```
oxy|doctype {
  display:block !important;
}
```

```

    color:blue !important;
    background-color:transparent !important;
}

```

- To match the processing instructions, you can use the *oxy/processing-instruction* selector:

```

oxy|processing-instruction {
    display:block !important;
    color:purple !important;
    background-color:transparent !important;
}

```

A processing instruction usually has a target and one or more pseudo attributes:

```
<?target_name data="b"?>
```

You can match a processing instruction with a particular target from the CSS using the construct:

```
oxy|processing-instruction[target_name]
```

You can also match the processing instructions having a certain target and pseudo attribute value like:

```
oxy|processing-instruction[target_name][data="b"]
```

- The XML comments display in Author mode can be changed using the *oxy/comment* selector:

```

oxy|comment {
    display:block !important;
    color:green !important;
    background-color:transparent !important;
}

```

- The *oxy/cdata* selector matches CDATA sections:

```

oxy|cdata{
    display:block !important;
    color:gray !important;
    background-color:transparent !important;
}

```

- The *oxy/entity* selector matches the entities content:

```

oxy|entity {
    display:morph !important;
    editable:false !important;
    color:orange !important;
    background-color:transparent !important;
}

```

- The *references to entities*, *XInclude*, and *DITA conrefs* and *conkeyrefs* are expanded by default in Author mode and the referenced content is displayed. The referenced resources are displayed inside the element or entity that refers to them.
 - You can use the *reference* property to customize the way these references are rendered in **Author** mode:

```

oxy|reference {
    border:1px solid gray !important;
}

```

In the **Author** mode, content is highlighted when parts of text contain:

- comments*.
- changes and *Track Changes* was active when the content was modified.

If this content is referenced, the **Author** mode does not display the highlighted areas in the new context. If you want to mark the existence of this comments and changes you can use the *oxy/reference[comments]*, *oxy/reference[changeTracking]*, and *oxy/reference[changeTracking][comments]* selectors.



Note: Two artificial attributes (*comments* and *changeTracking*) are set on the reference node, containing information about the number of comments and track changes in the content.

- The following example represents the customization of the reference fragments that contain comments:

```
oxy|reference[comments]:before {
    content: "Comments: " attr(comments) !important;
}
```

- To match reference fragments based on the fact that they contain change tracking inside, use the `oxy/reference[changeTracking]` selector.

```
oxy|reference[changeTracking]:before {
    content: "Change tracking: " attr(changeTracking) !important;
}
```

- Here is an example of how you can set a custom color to the reference containing both track changes and comments:

```
oxy|reference[changeTracking][comments]:before {
    content: "Change tracking: " attr(changeTracking) " and comments: " attr(comments) !important;
}
```

A sample document rendered using these rules:



Additional CSS Properties

Oxygen XML Editor Author offers an extension of the standard CSS properties suited for content editing.

Folding Elements: -oxy-foldable, -oxy-not-foldable-child and -oxy-folded properties

Oxygen XML Editor Author allows you to declare some elements to be *foldable* (collapsible). This is especially useful when working with large documents organized in logical blocks, editing a large DocBook article or book for instance. Oxygen XML Editor marks the foldable content with a small blue triangle. When you hover with your mouse pointer over this marker, a dotted line borders the collapsible content. The following contextual actions are available:

- **Close Other Folds**[Ctrl NumPad/ \(Command NumPad/ on OS X\)](#) - Folds all the elements except the current element.
- **Collapse Child Folds**[\(Ctrl+Decimal\)](#) - Folds the elements indented with one level inside the current element.
- **Expand Child Folds**[\(Ctrl+Equals\)](#)- Unfolds all child elements of the currently selected element.
- **Expand All**[\(Ctrl+NumPad+*\)](#) - Unfolds all elements in the current document.
- **Toggle Fold** - Toggles the state of the current fold.

To define the element whose content can be folded by the user, you must use the property: `-oxy-foldable:true;`. To define the elements that are folded by default, use the `-oxy-folded:true` property.

Note: The `-oxy-folded` property works in conjunction with the `-oxy-foldable` property. Thus, the `folded` property is ignored if the `-oxy-foldable` property is not set on the same element.

When collapsing an element, it is useful to keep some of its content visible, like a short description of the collapsed region. The property `-oxy-not-foldable-child` is used to identify the child element that is kept visible. It accepts as value an element name or a list of comma separated element names. The first child element from the XML document that appears in the list of element names will be identified as the not foldable child and displayed. If the element is marked as *foldable* (`-oxy-foldable:true;`) but it doesn't have the property `-oxy-not-foldable-child` or none of the specified non-foldable children exists, then the element is still foldable. In this case the element kept visible when folded will be the `before` pseudo-element.

Note: Deprecated properties `foldable`, `not-foldable-child`, and `folded` are also supported.

Folding DocBook Elements

All the elements below can have a `title` child element and are considered to be logical sections. You mark them as being *foldable* leaving the `title` element visible.

```
set,
book,
part,
reference,
chapter,
preface,
article,
sect1,
sect2,
sect3,
sect4,
section,
appendix,
figure,
example,
table {
    -oxy-foldable:true;
    -oxy-not-foldable-child: title;
}
```

Placeholders for empty elements: `-oxy-show-placeholder` and `-oxy-placeholder-content` properties

Oxygen XML Editor Author displays the element name as pseudo-content for empty elements, if the [Show placeholders for empty elements option](#) is enabled and there is no before or after content set in CSS for this type of element.

To control the displayed pseudo-content for empty elements, you can use the `-oxy-placeholder-content` CSS property.

The `-oxy-show-placeholder` property allows you to decide whether the placeholder must be shown. The possible values are:

- `always` - Always display placeholders.
- `default` - Always display placeholders if before or after content are not set in CSS.
- `inherit` - The placeholders are displayed according to [Show placeholders for empty elements](#) option (if before and after content is not declared).

Note: Deprecated properties `show-placeholder` and `placeholder-content` are also supported.

Read-only elements: -oxy-editable property

If you want to inhibit editing a certain element content, you can set the `-oxy-editable` (deprecated property `editable` is also supported) CSS property to `false`.

Display Elements: -oxy-morph value

Oxygen XML Editor Author allows you to specify that an element has an `-oxy-morph` display type (deprecated `morph` property is also supported), meaning that the element is inline if all its children are inline.

Let's suppose we have a **wrapper** XML element allowing users to set a number of attributes on all sub-elements. This element should have an inline or block behavior depending on the behavior of its child elements:

```
wrapper{
    display:-oxy-morph;
}
```

The whitespace property: -oxy-trim-when-ws-only value

Oxygen XML Editor Author allows you to set the `whitespace` property to `-oxy-trim-when-ws-only`, meaning that the leading and trailing whitespaces are removed.

The visibility property: -oxy-collapse-text

Oxygen XML Editor Author allows you to set the value of the `visibility` property to `-oxy-collapse-text`, meaning that the text content of that element is not rendered. If an element is marked as `-oxy-collapse-text` you are not able to position the caret inside it and edit it. The purpose of `-oxy-collapse-text` is to make the text value of an element editable only through a form control.

The text value of an XML element will be edited using a text field form control. In this case, we want the text content not to be directly present in the Author visual editing mode:

```
title{
    content: oxy_textfield(edit, '#text', columns, 40);
    visibility:-oxy-collapse-text;
}
```

Cyrillic Counters: list-style-type values -oxy-lower-cyrillic

Oxygen XML Editor Author allows you to set the value of the `list-style-type` property to `-oxy-lower-cyrillic-ru`, `-oxy-lower-cyrillic-uk`, `-oxy-upper-cyrillic-ru` or `-oxy-upper-cyrillic-uk`, meaning that you can have Russian and Ukrainian counters.

Counting list items with Cyrillic symbols:

```
li{
    display:list-item;
    list-style-type:-oxy-lower-cyrillic-ru;
}
```

The link property: link

Oxygen XML Editor Author allows you to declare some elements to be *links*. This is especially useful when working with many documents that reference each other. The links allow for an easy way to get from one document to another. Clicking on the link marker will open the referenced resource in an editor.

To define the element which should be considered a link, you must use the `link` property on the `before` or `after` pseudo element. The value of the property indicates the location of the linked resource. Since links are usually indicated by the value of an attribute in most cases it will have a value similar to `attr(href)`

DocBook Link Elements

All the elements below are defined to be links on the before pseudo element and their value is defined by the value of an attribute.

```
*[href]:before{
    link:attr(href);
    content: "Click " attr(href) " for opening" ;
}

ulink[url]:before{
    link:attr(url);
    content: "Click to open: " attr(url);
}

olink[targetdoc]:before{
    -oxy-link: attr(targetdoc);
    content: "Click to open: " attr(targetdoc);
}
```

Display Tag Markers: `-oxy-display-tags`

Oxygen XML Editor Author allows you to choose whether tag markers of an element should never be presented or the current display mode should be respected. This is especially useful when working with :before and :after pseudo-elements in which case the element range is already visually defined so the tag markers are redundant.

The property is named `-oxy-display-tags`, with the following possible values:

- *none* - Tags markers must not be presented regardless of the current [Display mode..](#)
- *default* - The tag markers will be created depending on the current [Display mode..](#)
- *inherit* - The value of the property is inherited from an ancestor element.

```
-oxy-display-tags
Value: none | default | inherit
Initial: default
Applies to: all nodes(comments, elements, CDATA, etc)
Inherited: false
Media: all
```

DocBook Para elements

In this example the **para** element from DocBook is using an :before and :after element so you don't want its tag markers to be visible.

```
para:before{
    content: "{";
}

para:after{
    content: "}";
}

para{
    -oxy-display-tags: none;
    display:block;
    margin: 0.5em 0;
}
```

Append Content Properties: `-oxy-append-content` and `-oxy-prepend-content`

The `-oxy-append-content` Property

This property appends the specified content to the content generated by other matching CSS rules of lesser specificity. Unlike the `content` property, where only the value from the rule with the greatest specificity is taken into account, the `-oxy-append-content` property adds content to that generated by the lesser specificity rules into a new compound content.

-oxy-append-content Example

```
element:before{
    content: "Hello";
}
element:before{
    -oxy-append-content: " World!";
}
```

The content shown before the element will be Hello World!.

The -oxy-prepend-content Property

Prepends the specified content to the content generated by other matching CSS rules of lesser specificity. Unlike the content property, where only the value from the rule with the greatest specificity is taken into account, the -oxy-prepend-content prepends content to that generated by the lesser specificity rules into a new compound content.

-oxy-prepend-content Example

```
element:before{
    content: "Hello!";
}
element:before{
    -oxy-prepend-content: "said: ";
}
element:before{
    -oxy-prepend-content: "I ";
}
```

The content shown before the element will be I said: Hello!.

Custom colors for element tags: -oxy-tags-color and -oxy-tags-background-color

By default Oxygen XML Editor does not display element tags. You can use the  **Partial Tags** button from the **Author** tool bar to control the amount of *displayed markup*.

To configure the default background and foreground colors of the tags, go to **Editor > Edit modes > Author**. The -oxy-tags-background-color and -oxy-tags-color properties allow you to control the background and foreground colors for any particular XML element.

```
para {
    -oxy-tags-color:white;
    -oxy-tags-background-color:green;
}
title {
    -oxy-tags-color:yellow;
    -oxy-tags-background-color:black;
}
```

Custom CSS Functions

The visual Author editing mode supports also a wide range of custom CSS extension functions.

The oxy_local-name() Function

The oxy_local-name() function evaluates the local name of the current node.

It does not have any arguments.

To insert as static text content before each element its local name, use this CSS selector:

```
*:before{
    content: oxy_local-name() ":";
```

The oxy_name() Function

The oxy_name() function evaluates the qualified name of the current node.

It does not have any arguments.

To insert as static text content before each element its qualified name, use this CSS selector:

```
*:before{
    content: oxy_name() " : ";
}
```

The oxy_url() Function

The `oxy_url()` function extends the standard CSS `url()` function by allowing you to specify additional relative path components (parameters `loc_1` to `loc_n`).

Oxygen XML Editor uses all these parameters to construct an absolute location. Note that any of the parameters that are passed to the function can be either relative or absolute locations. These locations can be expressed as String objects, functions, or editor variables (built-in or custom).

```
oxy_url( base_location , loc_1 , loc_2 )
```

base_location

String representing the base location. If not absolute, will be solved relative to the CSS file URL.

loc_1 ... loc_n (optional)

Strings representing relative location path components.

The following function receives String objects as input parameters:

```
oxy_url('http://www.oxygenxml.com/css/test.css', '../dir1/', 'dir2/dir3/',
'../../dir4/dir5/test.xml')
```

and returns:

```
'http://www.oxygenxml.com/dir1/dir4/dir5/test.xml'
```

The following function receives the result of the evaluation of two other functions as parameters:

```
image[href]{
    content:oxy_url(oxy_base-uri(), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
}
```

You can use the above example when you have image references and you want to see thumbnail images stored in the same folder.

The following function uses an editor variable as the first parameter to point to the Oxygen XML Editor installation location:

```
image[href] {
    content: oxy_url('${oxygenHome}', 'logo.png');
```

The oxy_base-uri() Function

The `oxy_base-uri()` function evaluates the base URL in the context of the current node.

It does not have any arguments and takes into account the `xml:base` context of the current node. See the [XML Base specification](#) for more details.

If you have image references but you want to see in the visual Author editing mode thumbnail images which reside in the same folder:

```
image[href]{
    content:oxy_url(oxy_base-uri(), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
}
```

The `oxy_parent-url()` Function

The `oxy_parent-url()` function evaluates the parent URL of an URL received as string.

`oxy_parent-url(URL)`

URL

The URL as string.

The `oxy_capitalize()` Function

The `oxy_capitalize` function capitalizes the first letter of the text received as argument.

`oxy_capitalize(text)`

text

The text for which the first letter will be capitalized.

To insert as static text content before each element its capitalized qualified name, use this CSS selector:

```
*:before{
    content: oxy_capitalize(oxy_name()) ":";
```

The `oxy_uppercase()` Function

The `oxy_uppercase()` function transforms to upper case the text received as argument.

`oxy_uppercase(text)`

text

The text to be capitalized.

To insert as static text content before each element its upper-cased qualified name, use this CSS selector:

```
*:before{
    content: oxy_uppercase(oxy_name()) ":";
```

The `oxy_lowercase()` Function

The `oxy_lowercase()` function transforms to lower case the text received as argument.

`oxy_lowercase(text)`

text

The text to be lower cased.

To insert as static text content before each element its lower-cased qualified name, use this CSS selector:

```
*:before{
    content: oxy_lowercase(oxy_name()) ":";
```

The `oxy_concat()` Function

The `oxy_concat()` function concatenates the received string arguments.

`oxy_concat(str_1 , str_2)`

str_1 ... str_n

The string arguments to be concatenated.

If an XML element has an attribute called **padding-left**:

```
<p padding-left="20">....
```

and you want to add a padding before it with that specific amount specified in the attribute value:

```
*[padding-left]{
    padding-left:oxy_concat(attr(padding-left), "px");
}
```

The oxy_replace() Function

The `oxy_replace` function is used to replace a string of text.

The `oxy_replace()` function has two signatures:

- `oxy_replace(text , target , replacement)`

This function replaces each substring of the text that matches the literal target string with the specified literal replacement string.

text

The text in which the replace will occur.

target

The target string to be replaced.

replacement

The string replacement.

- `oxy_replace(text , target , replacement , isRegExp)`

This function replaces each substring of the text that matches the target string with the specified replacement string.

text

The text in which the replace will occur.

target

The target string to be replaced.

replacement

The string replacement.

isRegExp

If *true* the target and replacement arguments are considered regular expressions, if *false* they are considered literal strings.

If you have image references but you want to see in the visual Author editing mode thumbnail images which reside in the same folder:

```
image[href]{
    content:oxy_url(oxy_base-uri()), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
}
```

The oxy_unparsed-entity-uri() Function

The `oxy_unparsed-entity-uri()` function returns the URI value of an unparsed entity name.

```
oxy_unparsed-entity-uri( unparsedEntityName )
```

unparsedEntityName

The name of an unparsed entity defined in the DTD.

This function can be useful to display images which are referenced with unparsed entity names.

CSS for displaying the image in Author for an `imagedata` with `entityref` to an unparsed entity

```
imagedata[entityref]{
  content: oxy_url(oxy_unparsed-entity-uri(attr(entityref)));
}
```

The `oxy_attributes()` Function

The `oxy_attributes()` function concatenates the attributes for an element and returns the serialization. `oxy_attributes()`

`oxy_attributes()`

For the following XML fragment:<element att1="x" xmlns:a="2" x=""" /> the CSS selector

```
element{
  content:oxy_attributes();
}
```

will display `att1="x" xmlns:a="2" x=" "`.

The `oxy_substring()` Function

The `oxy_substring()` function is used to return a string of text.

The `oxy_substring()` function has two signatures:

- `oxy_substring(text , startOffset)`

Returns a new string that is a substring of the original `text` string. It begins with the character at the specified index and extends to the end of `text` string.

`text`

The original string.

`startOffset`

The beginning index, inclusive

- `substring(text , startOffset , endOffset)`

Returns a new string that is a substring of the original `text` string. The substring begins at the specified `startOffset` and extends to the character at index `endOffset` - 1.

`text`

The original string.

`startOffset`

The beginning index, inclusive

`endOffset`

The ending index, exclusive.

`oxy_substring('abcd', 1)` returns the string 'bcd'.

`oxy_substring('abcd', 4)` returns an empty string.

`oxy_substring('abcd', 1, 3)` returns the string 'bc'.

If we want to display only part of an attribute's value, the part which comes before an **Appendix** string:

```
image[longdesc]{
    content: oxy_substring(attr(longdesc), 0, oxy_indexof(attr(longdesc), "Appendix"));
}
```

The oxy_getSomeText(text, length) Function

The `oxy_getSomeText(text, length)` function allows you to truncate a long string and to set a maximum number of displayed characters.

The following properties are supported:

- **text** - displays the actual text
- **length** - sets the maximum number of characters that are displayed
- **endsWithPoints** - specifies whether the truncated text ends with ellipsis

If an attribute value is very large we can trim its content before it is displayed as static content:

```
*[longdesc]:before{
    content: oxy_getSomeText(attr(longdesc), 200);
}
```

The oxy_indexof() Function

The `oxy_indexof()` function is used to define searches.

The `oxy_indexof()` function has two signatures:

- `oxy_indexof(text , toFind)`

Returns the index within **text** string of the first occurrence of the **toFind** substring.

text

Text to search in.

toFind

The searched substring.

- `oxy_indexof(text , toFind , fromOffset)`

Returns the index within **text** string of the first occurrence of the **toFind** substring. The search starts from **fromOffset** index.

text

Text to search in.

toFind

The searched substring.

fromOffset

The index from which to start the search.

```
oxy_indexof('abcd', 'bc') returns 1.  
oxy_indexof('abcdabc', 'bc', 2) returns 4.
```

If we want to display only part of an attribute's value, the part which comes before an **Appendix** string:

```
image[longdesc]{
    content: oxy_substring(attr(longdesc), 0, oxy_indexof(attr(longdesc), "Appendix"));
}
```

The oxy_lastindexof() Function

The `oxy_lastindexof()` function is used to define last occurrence searches.

The `oxy_lastindexof()` function has two signatures:

- `oxy_lastindexof(text , toFind)`

Returns the index within `text` string of the rightmost occurrence of the `toFind` substring.

text

Text to search in.

toFind

The searched substring.

- `oxy_lastindexof(text , toFind , fromOffset)`

The search starts from `fromOffset` index. Returns the index within `text` string of the last occurrence of the `toFind` substring, searching backwards starting from the `fromOffset` index.

text

Text to search in.

toFind

The searched substring.

fromOffset

The index from which to start the search backwards.

```
oxy_lastindexof('abcdabc', 'bc') returns 4.  
oxy_lastindexof('abcdbccdbc', 'bc', 2) returns 1.
```

If we want to display only part of an attribute's value, the part which comes before an **Appendix** string:

```
image[longdesc]{  
    content: oxy_substring(attr(longdesc), 0, oxy_lastindexof(attr(longdesc), "Appendix"));  
}
```

The `oxy_xpath()` Function

The `oxy_xpath()` function is used for XPath expressions.

The `oxy_xpath()` function has the following signature:

- `oxy_xpath(XPathExpression [, processChangeMarkers , value] [, evaluate , value])`

Evaluates the given XPath expression using Saxon 9 and returns the result. The parameters of the function are as follows:

- A required expression parameter, which is the XPath expression to be evaluated
- An optional `processChangeMarkers` parameter, followed by its value, which can be either `true` or `false` (default value). When you set the parameter to `true`, the function returns the resulting text with all the change markers accepted (`delete` changes are removed and `insert` changes are preserved).
- An optional `evaluate` parameter, followed by its value, which can have one of the following values:
 - `dynamic` - Evaluates the XPath each time there are changes in the document.
 - `dynamic-once` - Separately evaluates the XPath for each node that matches the CSS selector. It will not re-evaluate the expression when changes are made to other nodes in the document. This will lead to improved performance but the displayed content may not be updated to reflect the actual document content.
 - `static` - If the same XPath is evaluated on several nodes, the result for the first evaluation will be used for all other matches. Use this only if the XPath does not contain a relationship with the node on which the CSS property is evaluated. This will lead to improved performance but the static displayed content may not be updated to reflect the actual document content.



Note: The entities and `xi:include` sections are ignored when the XPath expressions are evaluated.

The following example counts the number of words from a paragraph (including tracked changes) and displays the result in front of it:

```
para:before{
    content:
        concat("/Number of words:",
            oxy_xpath(
                "count(tokenize(normalize-space(string-join(text(), '')), ' '))",
                processChangeMarkers,
                true),
            "/ ");
}
```

Form Controls

Oxygen XML Editor provides a variety of built-in form controls that allow users to interact with documents with familiar user interface objects.

Oxygen XML Editor provides the following built-in form controls:

- **Text Field** - A graphical user interface box that allows you to enter a single line of text.
- **Combo Box** - A graphical user interface object that can be a drop-down list or a combination of a drop-down list and a single-line text field.
- **Check Box** - A graphical user interface box that you can click to select or deselect a value.
- **Pop-up** - A contextual menu that provides quick access to various actions.
- **Button** - A graphical user interface object that performs a specific action.
- **Button Group** - A graphical user interface group of buttons (such as radio buttons) that perform specific actions.
- **Text Area** - A box that allows you to enter multiple lines of text.
- **URL Chooser** - A dialog box that allows you to select the location of local or remote resources.
- **Date Picker** - A form control object that allows you to select a date in a specified format.
- **HTML Content** - A graphical user interface box that is used for rendering HTML content.

For customization and backwards compatibility purposes, Oxygen XML Editor also supports a custom form control, the `oxy_editor()` function.

To watch our video demonstration in regards to form controls, go to http://oxygenxml.com/demo/Form_Controls.html.

The Text Field Form Control

The `oxy_textfield` built-in form control is used for entering a single line of text in a graphical user interface box. A text field may include optional content completion capabilities, used to present and edit the value of an attribute or an element.

The `oxy_textfield` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
 - `@attribute_name` - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
 - `#text` - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- `columns` - Controls the width of the form control. The unit size is the width of the `w` character.
- `width` - Specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).

- **fontInherit** - This value specifies whether the form control inherits its font from its parent element. The values of this property can be `true` or `false`. To make the pop-up form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- **visible** - Specifies whether or not the form control is visible. The possible values of this property are `true` (the form control is visible) and `false` (the form control is not visible).
- **values** - Specifies the values that populate the content completion list of proposals. If these values are not specified, they are collected from the associated schema.
- **tooltips** - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the `$(comma)` variable.
- **tooltip** - Specifies a tooltip to be displayed when you hover over the form control.
- **color** - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

Text Field Form Control

```
element {
    content: "Label: "
    oxy_textfield(
        edit, "@my_attr",
        values, "value1, value2"
        columns, 40);
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_textfield` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_textfield` code template.

The Combo Box Form Control

The `oxy_combobox` built-in form control is used for providing a graphical user interface object that is a drop-down list of proposed values. This form control can also be used for a combination of a drop-down list and an editable single-line text field.

The `oxy_combobox` form control supports the following properties:

- **edit** - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
 - **@attribute_name** - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
 - **#text** - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- **columns** - Controls the width of the form control. The unit size is the width of the `w` character.

- **width** - Specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px), and percentage (followed by the % character) length units. The **width** property takes precedence over the **columns** property (if the two are used together).
- **visible** - Specifies whether or not the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- **editable** - This property accepts the **true** and **false** values. In addition to a drop-down list, the **true** value also generates an editable text field box that allows you to insert other values than the proposed ones. The **false** value generates a drop-down list that only accepts the proposed values.
- **tooltips** - Associates tooltips to each value in the **values** property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the \${comma} variable.
- **values** - Specifies the values that populate the content completion list of proposals. If these values are not specified, they are collected from the associated schema..
- **fontInherit** - This value specifies whether the form control inherits its font from its parent element. The values of this property can be **true** or **false**.

 **Note:** To make the combo box form control inherit its font from its parent element, set the **fontInherit** property to **true**.

- **labels** - This property must have the same number of items as the **values** property. Each item provides a literal description of the items listed in the **values** property.

 **Note:** This property is only available for read-only combo boxes (the **editable** property is set to **false**).

- **color** - Specifies the foreground color of the form control. If the value of the **color** property is **inherit**, the form control has the same color as the element in which it is inserted.
- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

Combo Box Form Control

```
comboBox:before {
    content: "A combo box that edits an attribute value. The possible values are provided from
CSS:"
    oxy_combobox(
        edit, "@attribute",
        editable, true,
        values, "value1, value2, value3",
        labels, "Value no1, Value no2, Value no3");
}
```

 **Note:** You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a  symbol in the content complete list.

 **Tip:** To insert a sample of the `oxy_combobox` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_combobox` code template.

The Check Box Form Control

The `oxy_checkbox` built-in form control is used for a graphical user interface box that you can click to enable or disable an option. A single check-box or multiple check-boxes can be used to present and edit the value on an attribute or element.

The `oxy_checkbox` form control supports the following properties:

- **edit** - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
 - **@attribute_name** - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
 - **#text** - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- **resultSeparator** - If multiple check-boxes are used, the separator is used to compose the final result.
- **tooltips** - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the `${comma}` variable.
- **visible** - Specifies whether or not the form control is visible. The possible values of this property are `true` (the form control is visible) and `false` (the form control is not visible).
- **values** - Specifies the values that are committed when the check-boxes are selected. If these values are not specified in the CSS, they are collected from the associated XML Schema.
- **fontInherit** - This value specifies whether the form control inherits its font from its parent element. The values of this property can be `true` or `false`.



Note: To make the Check box form control inherit its font from its parent element, set the `fontInherit` property to `true`.

- **uncheckedValues** - Specifies the values that are committed when check-boxes are not selected.
- **labels** - This property must have the same number of items as the `values` property. Each item provides a literal description of the items listed in the `values` property.. If this property is not specified, the `values` property is used as the label.
- **columns** - Controls the width of the form control. The unit size is the width of the `w` character.
- **color** - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
  content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

Single Check-box Form Control

```
checkBox[attribute]:before {
  content: "A check box editor that edits a two valued attribute (On/Off).  

           The values are specified in the CSS:  

           oxy_checkbox(  

             edit, "@attribute",  

             values, "On",  

             uncheckedValues, "Off",  

             labels, "On/Off");
}
```

Multiple Check-boxes Form Control

```
multipleCheckBox[attribute]:before {
  content: "Multiple checkboxes editor that edits an attribute value.  

           Depending whether the check-box is selected a different value is committed:  

           oxy_checkbox(  

             edit, "@attribute",  

             values, "true, yes, on",
```

```

uncheckedValues, "false, no, off",
resultSeparator, ",",
labels, "Present, Working, Started");
}

```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_checkbox` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_checkbox` code template.

The Pop-up Form Control

The `oxy_popup` built-in form control is used to offer a contextual menu that provides quick access to various actions. A pop-up form control can display single or multiple selections.

The `oxy_popup` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
 - `@attribute_name` - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
 - `#text` - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- `rows` - This property specifies the number of rows that the form control presents.



Note: If the value of the `rows` property is not specified, the default value of `12` is used.

- `color` - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.



Note: This property is used for rendering in the **Author** mode.

- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (the form control is visible) and `false` (the form control is not visible).
- `tooltips` - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the `${ comma }` variable.
- `values` - Specifies the values that are committed when the check-boxes are selected. If these values are not specified in the CSS, they are collected from the associated XML Schema.
- `resultSeparator` - If multiple check-boxes are used, the separator is used to compose the final result.



Note: The value of the `resultSeparator` property cannot exceed one character.

- `selectionMode` - Specifies whether the form control allows the selection of a single value or multiple values. The predefined values of this property are `single` and `multiple`.
- `labels` - Specifies the label associated with each entry used for presentation. If this property is not specified, the `values` property is used instead.
- `columns` - Controls the width of the form control. The unit size is the width of the `w` character. This property is used for the visual representation of the form control.
- `width` - Specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).

- **rendererSort** - Allows you to sort the values rendered on the form control label. The possible values of this property are ascending and descending.
- **editorSort** - Allows you to sort the values rendered on the form control. The possible values of this property are ascending and descending.
- **rendererSeparator** - Defines a separator used when multiple values are rendered.
- **fontInherit** - This value specifies whether the form control inherits its font from its parent element. The values of this property can be **true** or **false**.



Note: To make the Pop-up form control inherit its font from its parent element, set the **fontInherit** property to **true**.



Tip: In the subsequent example, the value of the **fontInherit** property is **true**, which means the pop-up form control inherits the font size of **30px** from the **font-size** property.

- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

Pop-up Form Control

```
popupWithMultipleSelection:before {
    content: " This editor edits an attribute value. The possible values are specified
inside the CSS: "
    oxy_popup(
        edit, "@attribute",
        values, "value1, value2, value3, value4, value5",
        labels, "Value no1, Value no2, Value no3, Value no4, Value no5",
        resultSeparator, "/",
        columns, 10,
        selectionMode, "multiple",
        fontInherit, true);
    font-size:30px;
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_popup` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_popup` code template.

The Button Form Control

The `oxy_button` built-in form control is used for graphical user interface objects that invokes a custom Author action (defined in the associated Document Type) referencing it by its ID, or directly in the CSS.

The `oxy_button` form control supports the following properties:

- **actionContext** - Specifies the context in which the action associated with the form control is executed. Its possible values are `element` and `caret`. If you select the `element` value, the context is the element that holds the form control. If you select the `caret` value, the action is invoked at the caret location. If the caret is not inside the element that holds the form control, the `element` value is selected automatically.
- **fontInherit** - This value specifies whether the form control inherits its font from its parent element. The values of this property can be **true** or **false**. To make the `button` form control inherit its font from its parent element, set the **fontInherit** property to **true**.

- color - Specifies the foreground color of the form control. If the value of the color property is inherit, the form control has the same color as the element in which it is inserted.
- actionID - The ID of the action, specified in the associated *document type framework*, that is invoked when you click the button.



Note: The element that contains the form control represents the context where the action is invoked.

- action - Defines an action directly, rather than using the actionID parameter to reference an action from the associated *document type framework*. This property is defined using the *oxy_action function*.

```
oxy_button(action, oxy_action(
    name, 'Insert',
    description, 'Insert an element after the current one',
    icon, url('insert.png'),
    operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
    arg-fragment, '<element>${caret}</element>',
    arg-insertLocation, '.',
    arg-insertPosition, 'After'
)
```



Tip: A code template is available to make it easy to add the *oxy_action* function.

- visible - Specifies whether or not the form control is visible. The possible values of this property are true (the form control is visible) and false (the form control is not visible).
- transparent - Flattens the aspect of the button form control, removing its border and background.
- showText - Specifies if the action text should be displayed on the button form control. If this property is missing then the button displays the icon only if it is available, or the text if the icon is not available. The values of this property can be true or false.

```
element {
    content: oxy_button(actionID, 'remove.attribute', showText, true);
}
```

- showIcon - Specifies if the action icon should be displayed on the button form control. If this property is missing then the button displays the icon only if it is available, or the text if the icon is not available. The values of this property can be true or false.

```
element {
    content: oxy_button(actionID, 'remove.attribute', showIcon, true);
}
```

- enableInReadOnlyContext - To enable *button form controls* or *groups of buttons form controls* this property needs to be set to true. This property can be used to specify areas as *read-only* (by setting the -oxy-editable property to false). This is useful when you want to execute an action that does not modify the context.
- hoverPseudoclassName - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

Button Form Control

```
button:before {
    content: "Label:"
    oxy_button(
        /* This action is declared in the document type associated with the XML document.
    */
        actionID, "insert.popupWithMultipleSelection");
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_button` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_button` code template. Also, an `oxy_button_in_place_action` code template is available that inserts an `oxy_button` function that includes an `action` parameter.

The Button Group Form Control

The `oxy_buttonGroup` built-in form control is used for a graphical user interface group of buttons that invokes one of several custom Author actions (defined in the associated Document Type) referencing it by its ID, or directly in the CSS.

The `oxy_buttonGroup` form control supports the following properties:

- `actionIDs` - The IDs of the actions that will be presented in the group of buttons.
- `actionID` - The ID of the action, specified in the associated *document type framework*, that is invoked when you click the button.



Note: The element that contains the form control represents the context where the action is invoked.

- `action_list` - Defines a list of actions directly, rather than using the `actionID` parameter to reference actions from the associated *document type framework*. This property is defined using the *oxy_action_list function*.

```
oxy_buttonGroup(
    label, 'A group of actions',
    icon, url('http://www.oxygenxml.com/img/icn_oxy20.png'),
    actions,
    oxy_action_list(
        oxy_action(
            name, 'Insert',
            description, 'Insert an element after the current one',
            operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
            arg-fragment, '<element></element>',
            arg-insertLocation, ',',
            arg-insertPosition, 'After'
        ),
        oxy_action(
            name, 'Delete',
            description, 'Deletes the current element',
            operation, 'ro.sync.ecss.extensions.commons.operations.DeleteElementOperation'
        )
    )
)
```



Tip: A code template is available to make it easy to add the `oxy_action_list` function.

- `label` - Specifies the label to be displayed on the button.
- `icon` - The path to the icon to be displayed on the button.
- `actionContext` - Specifies the context in which the action associated with the form control is executed. Its possible values are `element` and `caret`. If you select the `element` value, the context is the element that holds the form control. If you select the `caret` value, the action is invoked at the caret location. If the caret is not inside the element that holds the form control, the `element` value is selected automatically.
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (the form control is visible) and `false` (the form control is not visible).
- `actionStyle` - Specifies what to display for an action in the form control. The values of this property can be `text`, `icon`, or `both`.
- `tooltip` - Specifies a tooltip to be displayed when you hover over the form control.
- `transparent` - Makes the button transparent without any borders or background colors. The values of this property can be `true` or `false`.
- `fontInherit` - This value specifies whether the form control inherits its font from its parent element. The values of this property can be `true` or `false`.



Note: To make the form control inherit its font from its parent element, set the **fontInherit** property to `true`.

- `enableInReadOnlyContext` - To enable *button form controls* or *groups of buttons form controls* this property needs to be set to `true`. This property can be used to specify areas as *read-only* (by setting the `-oxy-editable` property to `false`). This is useful when you want to execute an action that does not modify the context.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

The Button Group Form Control

```
buttongroup:before {
    content:
        oxy_label(text, "Button Group:", width, 150px, text-align, left)
        oxy_buttonGroup(
            label, 'A group of actions',
            /* The action IDs are declared in the document type associated with the XML
            document. */
            actionIDs, "insert.popupWithMultipleSelection,insert.popupWithSingleSelection",
            actionStyle, "both");
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_buttonGroup` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_buttonGroup` code template. Also, an `oxy_buttonGroup_in_place_action` code template is available that inserts an `oxy_buttonGroup` function that includes an `oxy_action_list` function.

The Text Area Form Control

The `oxy_textArea` built-in form control is used for entering multiple lines of text in a graphical user interface box. A text area may include optional syntax highlight capabilities to present the form control.

The `oxy_textArea` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
 - `@attribute_name` - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
 - `#text` - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- `columns` - Controls the width of the form control. The unit size is the width of the `w` character.
- `width` - Specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `fontInherit` - This value specifies whether the form control inherits its font from its parent element. The values of this property can be `true` or `false`.

- **visible** - Specifies whether or not the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- **rows** - This property specifies the number of rows that the form control presents. If the form control has more lines, you are able to scroll and see them all.
- **contentType** - Specifies the type of content for which the form control offers syntax highlighting. The following values are supported: `text/css`; `text/shell`; `text/cc`; `text/xquery`; `text/xml`; `text/python`; `text/xsd`; `text/c`; `text>xpath`; `text/javascript`; `text/xsl`; `text/wsdl`; `text/html`; `text>xproc`; `text/properties`; `text/sql`; `text/rng`; `text/sch`; `text/json`; `text/perl`; `text/php`; `text/java`; `text/batch`; `text/rnc`; `text/dtd`; `text/nvdl`; `text/plain`.
- **indentOnTab** - Specifies the behaviour of the **Tab** key. If the value of this property is set to **true**, the **Tab** key inserts characters. If it is set to **false**, **Tab** is used for navigation, jumping to the next editable position in the document.
- The `white-space` CSS property influences the value that you edit, as well as the form control size:
 - **pre** - The whitespaces and new lines of the value are preserved and edited. If the **rows** and **columns** properties are not specified, the form control calculates its size on its own so that all the text is visible.
 - **pre-wrap** - The long lines are wrapped to avoid horizontal scrolling.



Note: The `rows` and `columns` properties must be specified. If these are not specified, the form control considers the value to be `pre`.

- **normal** - The white spaces and new lines are normalized.
- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
  content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

The following example presents a text area with CSS syntax highlighting that calculates its own dimension, and a second one with XML syntax highlighting with defined dimension.

```
textArea {
  visibility: -oxy-collapse-text;
  white-space: pre;
}

textArea[language="CSS"]:before {
  content: oxy_textArea(
    edit, '#text',
    contentType, 'text/css');
}

textArea[language="XML"]:before {
  content: oxy_textArea(
    edit, '#text',
    contentType, 'text/xml',
    rows, 10,
    columns, 30);
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_textArea` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_textArea` code template.

The URL Chooser Form Control

The `oxy_urlChooser` built-in form control is used for a dialog box that allows you to select the location of local or remote resources. The inserted reference is made relative to the URL of the currently opened editor.

The `oxy_urlChooser` editor supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
 - `@attribute_name` - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
 - `#text` - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- `columns` - Controls the width of the form control. The unit size is the width of the `w` character.
- `width` - Specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `color` - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (the form control is visible) and `false` (the form control is not visible).
- `fontInherit` - This value specifies whether the form control inherits its font from its parent element. The values of this property can be `true` or `false`.
- `fileFilter` - string value that holds comma-separated file extensions. The URL chooser uses these extensions to filter the displayed files. A value such as "`jpg, png, gif`" is mapped to a single filter that will display all `jpg`, `png`, and `gif` files.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
  content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

URL Chooser Form Control

```
urlChooser[file]:before {
  content: "An URL chooser editor that allows browsing for a URL. The selected URL is made
relative to the currently edited file."
  oxy_urlChooser(
    edit, "@file",
    columns 25);
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_urlChooser` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_urlChooser` code template.

The Date Picker Form Control

The `oxy_datePicker` built-in form control is used for offering a text field with a calendar browser that allows to choose a certain date in a specified format.

The oxy_datePicker form control supports the following properties:

- **edit** - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:

- **@attribute_name** - The name of the attribute whose value is being edited. If the attribute is in a namespace, the value of the property must be a *QName* and the CSS must have a namespace declaration for the prefix.
- **#text** - Specifies that the presented/edited value is the simple text value of an element.



Note: You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the `oxy_editor` function specifies.

- **columns** - Controls the width of the form control. The unit size is the width of the `w` character.
- **width** - Specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- **color** - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- **format** - This property specifies the format of the inserted date. The pattern value must be a valid Java date (or date-time) format. If missing, the type of the date is determined from the associated schema.
- **visible** - Specifies whether or not the form control is visible. The possible values of this property are `true` (the form control is visible) and `false` (the form control is not visible).
- **validateInput** - Specifies if the form control is validated. If you introduce a date that does not respect the format, the `datePicker` form control is rendered with a red foreground. By default, the input is validated. To disable the validation, set this property to `false`.
- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

Date Picker Form Control

```
date {
    content:
        oxy_label(text, "Date time attribute with format defined in CSS: ", width, 300px)
        oxy_datePicker(
            columns, 16,
            edit, "@attribute",
            format, "yyyy-MM-dd");
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the `oxy_datePicker` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the oxy_datePicker code template.

The HTML Content Form Control

The `oxy_htmlContent` built-in form control is used for rendering HTML content. This HTML content is displayed as a graphical element shaped as a box. The shape of the box is determined by a given width and the height is computed based upon the length of the text.

The `oxy_htmlContent` form control supports the following properties:

- **href** - The absolute or relative location of a resource. The resource needs to be a well-formed HTML file.
- **id** - The unique identifier of an item. This is a **div** element that has a unique **id** and is a child of the **body** element. The **div** element is the container of the HTML content to be rendered by the form control.
- **content** - An alternative to the **href** and **id** pair of elements. It provides the HTML content that will be displayed in the form control.
- **width** - Specifies the width of the content area using relative (**em**, **ex**), absolute (**in**, **cm**, **mm**, **pt**, **pc**, **px**), and percentage (followed by the **%** character) length units. The **width** property takes precedence over the **columns** property (if the two are used together).
- **hoverPseudoclassName** - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo class. When you hover over the form control, the specified pseudo class will be set on the element that contains the form control.

```
p:before {
    content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
    border: 1px solid red;
}
```

You can customize the style of the content using CSS that is either referenced by the file identified by the **href** property or is defined in-line. If you change the HTML content or CSS and you want your changes to be reflected in the XML that renders the form control, then you need to refresh the XML file. If the HTML does not have an associated style, then a default text and background color will be applied.

In the following example, the form control collects the content from the *p_description* **div** element found in the *descriptions.html* file. The box is 400 pixels wide and is displayed before a paragraph identified by the **intro_id** attribute value.

```
p#intro_id:before {
    content:
        oxy_htmlContent(
            href, "descriptions.html",
            id, "p_description",
            width, 400px);
}
```

An alternative example, using the **content** property:

```
p#intro_id:before {
    content:
        oxy_htmlContent(
            content, "<div style='font-weight:bold;'>My content</div>",
            width, 400px);
}
```



Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.



Tip: To insert a sample of the **oxy_htmlContent** form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the **oxy_htmlContent** code template.

Implementing Custom Form Controls

If the built-in form controls are not sufficient for your needs, you can implement custom form controls in Java.

You can specify them using the following properties:

- **rendererClassName** - the name of the class that draws the edited value. It must be an implementation of **ro.sync.ecss.extensions.api.editor.InplaceRenderer**. The renderer has to be a **SWING** implementation and can be used both in the standalone and Eclipse distributions.

- **swingEditorClassName** - you can use this property for the standalone (**Swing**-based) distribution to specify the name of the class used for editing. It is a **Swing** implementation of `ro.sync.ecss.extensions.api.editor.InplaceEditor`.
- **swtEditorClassName** - you can use this property for the Eclipse plug-in distribution to specify the name of the class used for editing. It is a **SWT** implementation of the `ro.sync.ecss.extensions.api.editor.InplaceEditor`.
- **classpath** - you can use this property to specify the location of the classes used for a custom form control. The value of the **classpath** property is an enumeration of URLs separated by comma.
- **edit** - if your form control edits the value of an attribute or the text value of an element, you can use the `@attribute_name` and `#text` predefined values and oxygen will perform the commit logic by itself. You can use the **custom** value to perform the commit logic yourself.

Custom Form Control Implementation

Sample Java code for a custom combo box form control implementation that inserts an XML element in the content when the editing stops:

```
public class ComboBoxEditor extends AbstractInplaceEditor {
    /**
     * @see ro.sync.ecss.extensions.api.editor.InplaceEditor#stopEditing()
     */
    @Override
    public void stopEditing() {
        Runnable customCommit = new Runnable() {
            @Override
            public void run() {
                AuthorDocumentController documentController =
context.getAuthorAccess().getDocumentController();
                documentController.insertXMLFragment( "<custom/>", offset);
            }
        };
        EditingEvent event = new EditingEvent(customCommit, true);
        fireEditingStopped(event);
    }
}
```

If the custom form control is intended to work in the Oxygen XML Editor standalone distribution, the declaration of **swtEditorClassName** is not required. The *renderer* (the class that draws the value) and the *editor* (the class that edits the value) have different properties because you can present a value in one way and edit it in another.

The custom form controls can use any of the predefined properties of the `oxy_editor` function, as well as specified custom properties. This is an example of how to specify a custom form control:

```
myElement {
    content: oxy_editor(
        rendererClassName, "com.custom.editors.CustomRenderer",
        swingEditorClassName, "com.custom.editors.SwingCustomEditor",
        swtEditorClassName, "com.custom.editors.SwtCustomEditor",
        edit, "@my_attr",
        customProperty1, "customValue1",
        customProperty2, "customValue2"
    )
}
```



Note: Add these custom **Java** implementations in the *classpath* of the document type associated with the document you are editing. To get you started, the **Java** sources for the `SimpleURLChooserEditor` are available in the [Oxygen SDK](#).

The `oxy_editor` function can receive other functions as parameters for obtaining complex behaviors.

The following example shows how the combo box editor can obtain its values from the current XML file by calling the `oxy_xpath` function:

```
link:before{
    content: "Managed by:" oxy_editor(
        type, combo,
```

```
edit, "@manager",
values, oxy_xpath('string-join(//@id , "," )'));
```

Editing Processing Instructions Using Form Controls

Oxygen XML Editor allows you to edit *processing instructions*, *comments*, and *CDATA* by using the built-in editors.

Oxygen XML Editor allows you to edit *processing instructions*, *comments*, and *CDATA* by using the built-in editors.

 **Note:** You can edit both the content and the attribute value from a *processing instruction*.

Editing an Attribute from a Processing Instruction

PI content

```
<?pi_target attr="val"?>
```

CSS

```
oxy|processing-instruction:before {
    display:inline;
    content:
        "EDIT attribute: " oxy_textfield(edit, '@attr', columns, 15);
    visibility:visible;
}
oxy|processing-instruction{
    visibility:-oxy-collapse-text;
}
```

The `oxy_action()` Function

The `oxy_action()` function allows you to define actions directly in the CSS, rather than referencing them from the associated framework.

The `oxy_action()` function is used from [the `oxy_button\(\)` function](#).

The arguments received by the `oxy_action()` function are a list of properties that define an action. The following properties are supported:

- `name` - The name of the action. It will be displayed as the label for the button or menu item.
- `description` (optional) - A short description with details about the result of the action.
- `icon` (optional) - A path relative to the CSS pointing to an image (the icon for the action). The path can point to resources that are packed in Oxygen XML Editor (`oxygen.jar`) by starting its value with / (for example, `/images/Remove16.png`). It can also be expressed as [editor variables](#).
- `operation` - The name of the Java class implementing the `ro.sync.ecss.extensions.api.AuthorOperation` interface. There is also a variety of [predefined operations](#) that can be used.



Note: If the name of the operation specified in the CSS is not qualified (has no Java package name), then it is considered to be one of the built-in Oxygen XML Editor operations from `ro.sync.ecss.extensions.commons.operations` package. If the class is not found in this package, then it will be loaded using the specified name.

- `arg-<string>` - All arguments with the `arg-` prefix are passed to the operation (the string that follows the `arg-` prefix is passed).
- `ID` - (optional) - The ID of the action from the framework. If this is specified, all others parameters are disregarded.

```
oxy_button(
    action, oxy_action(
        name, 'Insert',
        description, 'Insert an element after the current one',
        icon, url('insert.png'),
```

```

        operation,
        'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
        arg-fragment, '<element>${caret}</element>',
        arg-insertLocation, '.',
        arg-insertPosition, 'After'),
        showIcon, true)
    
```



Tip: A code template is available to make it easy to add the `oxy_action` function with the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `.oxy_action` code template..

The `oxy_action_list()` Function

The `oxy_action_list()` function allows you to define a list of actions directly in the CSS, rather than referencing them from the associated framework.

The `oxy_action_list()` function is used from [the `oxy_buttonGroup\(\)` function](#).

The arguments received by the `oxy_action_list()` function are a list of actions that are defined with [the `oxy_action\(\)` function](#). The following properties are supported in the `oxy_action_list()` function:

- name - The name of the action. It will be displayed as the label for the button or menu item.
- description (optional) - A short description with details about the result of the action.
- icon (optional) - A path relative to the CSS pointing to an image (the icon for the action). The path can point to resources that are packed in Oxygen XML Editor (`oxygen.jar`) by starting its value with / (for example, `/images/Remove16.png`). It can also be expressed as [editor variables](#).
- operation - The name of the Java class implementing the [ro.sync.ecss.extensions.api.AuthorOperation](#) interface. There is also a variety of [predefined operations](#) that can be used.



Note: If the name of the operation specified in the CSS is not qualified (has no Java package name), then it is considered to be one of the built-in Oxygen XML Editor operations from `ro.sync.ecss.extensions.commons.operations` package. If the class is not found in this package, then it will be loaded using the specified name.

- `arg-<string>` - All arguments with the `arg-` prefix are passed to the operation (the string that follows the `arg-` prefix is passed).
- ID - (optional) - The ID of the action from the framework. If this is specified, all others parameters are disregarded.

```

oxy_action_list(
    oxy_action(
        name, 'Insert',
        description, 'Insert an element after the current one',
        operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
        arg-fragment, '<element></element>',
        arg-insertLocation, '.',
        arg-insertPosition, 'After'
    ),
    oxy_action(
        name, 'Delete',
        description, 'Deletes the current element',
        operation, 'ro.sync.ecss.extensions.commons.operations.DeleteElementOperation'
    )
)
    
```



Tip: A code template is available to make it easy to add the `oxy_action_list` function with the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `.oxy_action_list` code template.

The `oxy_label()` Function

The `oxy_label()` function can be used in conjunction with the CSS `content` property to change the style of generated text.

The arguments of the function are *property name - property value* pairs. The following properties are supported:

- **text** - This property specifies the built-in form control you are using.
- **width** - Specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px), and percentage (followed by the % character) length units. The width property takes precedence over the columns property (if the two are used together).
- **color** - Specifies the foreground color of the form control. If the value of the color property is inherit, the form control has the same color as the element in which it is inserted.
- **background-color** - Specifies the background color of the form control. If the value of the background-color property is inherit, the form control has the same color as the element in which it is inserted.
- **styles** - Specifies styles for the form control. The values of this property are a set of CSS properties:
 - font-weight, font-size, font-style, font
 - text-align, text-decoration
 - width
 - color, background-color
 - link - For more information on this property see [the link property section](#).

```
element{
  content: oxy_label(text, "Label Text", styles,
    "font-size:2em;color:red;link:attr(href');");
}
```

If the text from an oxy_label() function contains new lines, for example oxy_label(text, 'LINE1\\A LINE2', width, 100px), the text is split in two. Each of the two new lines has the specified width of 100 pixels.

 **Note:** The text is split after \A, which represents a new line character.

You can use the [oxy_editor\(\)](#) and [oxy_label\(\)](#) functions together to create a form control based layout.

Let's say we want to edit two attributes on a single element using form controls on separate lines:

```
person:before {
  content: "Name:*" oxy_textfield(edit, '@name', columns, 20) "\A Address:" oxy_textfield(edit,
  '@address', columns, 20)
}
```

We can use [oxy_label\(\)](#) if we want only the **Name** label to be bold and also to properly align the two controls:

```
person:before {
  content: oxy_label(text, "Name:", styles, "font-weight:bold;width:200px") oxy_textfield(edit,
  '@name', columns, 20) "\A "
    oxy_label(text, "Address:", styles, "width:200px") oxy_textfield(edit, '@address',
  columns, 20)
}
```

 **Tip:** A code template is available to make it easy to add the [oxy_label](#) function with the **Content Completion Assistant** by pressing [Ctrl Space \(Command Space on OS X\)](#) and select the [.oxy_label](#) code template..

The [oxy_link-text\(\)](#) Function

You can use the [oxy_link-text\(\)](#) function on the CSS content property to obtain a text description from the source of a reference.

By default, the [oxy_link-text\(\)](#) function resolves DITA and DocBook references. For further details about how you can also extend this functionality to other frameworks, go to [Configuring an Extensions Bundle](#).

DITA Support

For DITA, the [oxy_link-text\(\)](#) function resolves the `xref` element and the elements that have a `keyref` attribute. The text description is the same as the one presented in the final output for those elements. If you use this function for

a `topicref` element that has the `navtitle` and `locktitle` attributes set, the function returns the value of the `navtitle` attribute.

DocBook Support

For DocBook, the `oxy_link-text()` function resolves the `xref` element that defines a link in the same document. The text description is the same as the one presented in the final output for those elements.

For the following XML and associated CSS fragments the `oxy_link-text()` function is resolved to the value of the `xreflabel` attribute.

```
<para><code id="para.id" xreflabel="The reference label">my code</code></para>
<para><xref linkend="para.id"/></para>

xref {
    content: oxy_link-text();
}
```

If the text from the target cannot be extracted (for instance, if the `href` is not valid), you can use an optional argument to display fallback text.

```
*[class~="map/topicref"]:before{
    content: oxy_link-text("Cannot find the topic reference");
    link:attr(href);
}
```

The `oxy_unescapeURLValue(string)` Function

The `oxy_unescapeURLValue()` function returns the unescaped value of an URL-like string given as a parameter.

For example if the value contains `%20` it will be converted to a simple space character.

```
oxy_unescapeURLValue( "http://www.example.com/a%20simple%20example.html" )
returns the http://www.example.com/a simple example.html value.
```

Arithmetic Functions

Arithmetic Functions are supported.

You can use any of the arithmetic functions implemented in the `java.lang.Math` class:

<http://download.oracle.com/javase/6/docs/api/java/lang/Math.html>.

In addition to that, the following functions are available:

Syntax	Details
<code>oxy_add(param1, ..., paramN, 'returnType')</code>	Adds the values of all parameters from <code>param1</code> to <code>paramN</code> .
<code>oxy_subtract(param1, ..., paramN, 'returnType')</code>	Subtracts the values of parameters <code>param2</code> to <code>paramN</code> from <code>param1</code> .
<code>oxy_multiply(param1, ..., paramN, 'returnType')</code>	Multiplies the values of parameters from <code>param1</code> to <code>paramN</code> .
<code>oxy_divide(param1, param2, 'returnType')</code>	Performs the division of <code>param1</code> to <code>param2</code> .
<code>oxy_modulo(param1, param2, 'returnType')</code>	Returns the remainder of the division of <code>param1</code> to <code>param2</code> .



Note: The `returnType` can be '`integer`', '`number`', or any of the supported CSS measuring types.

If we have an image with **width** and **height** specified on it we can compute the number of pixels on it:

```
image:before{
    content: "Number of pixels: " oxy_multiply(attr(width), attr(height), "px");
}
```

Custom CSS Pseudo-classes

You can set your custom CSS pseudo-classes on the nodes from the *AuthorDocument* model. These are similar to the normal XML attributes, with the important difference that they are not serialized, and by changing them the document does not create undo and redo edits - the document is considered unmodified. You can use custom pseudo-classes for changing the style of an element (and its children) without altering the document.

In Oxygen XML Editor they are used to hide/show the `colspec` elements from CALS tables. To take a look at the implementation, see:

1. `[OXYGEN_DIR]/frameworks/docbook/css/cals_table.css` (Search for `-oxy-visible-colspecs`)
2. The definition of action `table.toggle.colspec` from the DocBook 4 framework makes use of the pre-defined [TogglePseudoClassOperation](#) Author operation.

Here are some examples:

Controlling the visibility of a section using a pseudo-class

You can use a non standard (custom) pseudo-class to impose a style change on a specific element. For instance you can have CSS styles matching the custom pseudo-class `access-control-user`, like the one below:

```
section {
    display:none;
}

section:access-control-user {
    display:block;
}
```

By setting the pseudo-class `access-control-user`, the element `section` will become visible by matching the second CSS selector.

Coloring the elements over which the caret was placed

```
*:caret-visited {
    color:red;
}
```

You could create an [AuthorCaretListener](#) that sets the `caret-visited` pseudo-class to the element at the caret location. The effect will be that all the elements traversed by the caret become red.

The API you can use from the caret listener:

```
ro.sync.ecss.extensions.api.AuthorDocumentController#setPseudoClass(java.lang.String,
ro.sync.ecss.extensions.api.node.AuthorElement)
ro.sync.ecss.extensions.api.AuthorDocumentController#removePseudoClass(java.lang.String,
ro.sync.ecss.extensions.api.node.AuthorElement)
```

Pre-defined [AuthorOperations](#)s can be used directly in your framework ("Author/Actions") to work with custom pseudo classes:

1. [TogglePseudoClassOperation](#)
2. [SetPseudoClassOperation](#)
3. [RemovePseudoClassOperation](#)

Built in CSS Stylesheet

When Oxygen XML Editor renders content in the **Author** mode, it adds built-in CSS selectors (in addition to the CSS stylesheets linked in the XML or specified in the document type associated to the XML document). These built-in CSS

selectors are processed before all other CSS content, but they can be overwritten in case the CSS developer wants to modify a default behavior.

List of CSS Selector Contributed by Oxygen XML Editor

```

@namespace oxy "http://www.oxygenxml.com/extensions/author";
@namespace xi "http://www.w3.org/2001/XInclude";
@namespace xlink "http://www.w3.org/1999/xlink";
@namespace svg "http://www.w3.org/2000/svg";
@namespace mml "http://www.w3.org/1998/Math/MathML";

oxy|document {
    display:block !important;
}

oxy|cdata {
    display:-oxy-morph !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|processing-instruction {
    display:block !important;
    color: rgb(139, 38, 201) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|comment {
    display:-oxy-morph !important;
    color: rgb(0, 100, 0) !important;
    background-color:rgb(255, 255, 210) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|reference:before,
oxy|entity[href]:before{
    link: attr(href) !important;
    text-decoration: underline !important;
    color: navy !important;

    margin: 2px !important;
    padding: 0px !important;
}

oxy|reference:before {
    display: -oxy-morph !important;
    content: url(..../images/editContent.gif) !important;
}

oxy|entity[href]:before{
    display: -oxy-morph !important;
    content: url(..../images/editContent.gif) !important;
}

oxy|reference,
oxy|entity {
    -oxy-editable:false !important;
    background-color: rgb(240, 240, 240) !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|reference {
    display:-oxy-morph !important;
    /*EXM-28674 No need to present tags for these artificial references.*/
    -oxy-display-tags: none;
}

oxy|entity {
    display:-oxy-morph !important;
}

oxy|entity[href] {
    border: 1px solid rgb(175, 175, 175) !important;
    padding: 0.2em !important;
}

xi|include {
    display:-oxy-morph !important;
}

```

```

    margin-bottom: 0.5em !important;
    padding: 2px !important;
}
xi|include:before,
xi|include:after{
    display:inline !important;
    background-color:inherit !important;
    color:#444444 !important;
    font-weight:bold !important;
}

xi|include:before {
    content:url(..../images/link.gif) attr(href) !important;
    link: attr(href) !important;
}
xi|include[xpointer]:before {
    content:url(..../images/link.gif) attr(href) " " attr(xpointer) !important;
    link: oxy_concat(attr(href), "#", attr(xpointer)) !important;
}

xi|fallback {
    display:-oxy-morph !important;
    margin: 2px !important;
    border: 1px solid #CB0039 !important;
}

xi|fallback:before {
    display:-oxy-morph !important;
    content:"XInclude fallback: " !important;
    color:#CB0039 !important;
}

oxy|doctype {
    display:block !important;
    background-color: transparent !important;
    color:blue !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 2px !important;
}

oxy|error {
    display:-oxy-morph !important;
    -oxy-editable:false !important;
    white-space:pre !important;
    color: rgb(178, 0, 0) !important;
    font-weight:bold !important;
}

oxy|error:before {
    content:url(..../images/ReferenceError.png) !important;
}

*[xlink|href]:before {
    content:url(..../images/link.gif);
    link: attr(xlink|href) !important;
}

/*No direct display of the MathML and SVG images.*/
svg|svg{
    display:inline !important;
    white-space: -oxy-trim-when-ws-only;
}
/*EXM-28827 SVG can contain more than one namespace in it*/
svg|svg * {
    display:none !important;
    white-space:normal;
}

mml|math{
    display:inline !important;
    white-space: -oxy-trim-when-ws-only;
}
mml|math mml|*{
    display:none !important;
    white-space: normal;
}

/*Text direction attributes*/
*[dir='rtl'] { direction:rtl; unicode-bidi:embed; }
*[dir='rl0'] { direction:rtl; unicode-bidi:bidi-override; }

*[dir='ltr'] { direction:ltr; unicode-bidi:embed; }
*[dir='lro'] { direction:ltr; unicode-bidi:bidi-override; }

```

To show all entities in the **Author** mode as transparent, without that grayed-out background, first define in your CSS after all imports the namespace:

```
@namespace oxy "http://www.oxygenxml.com/extensions/author";
```

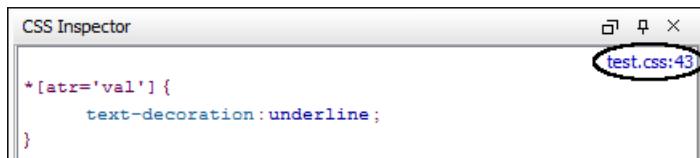
and then add the following selector:

```
oxy|entity {
    background-color: inherit !important;
}
```

Debugging CSS Stylesheets

To assist you with debugging and customizing CSS stylesheets the **Author** mode includes a **CSS Inspector view** to examine the CSS rules that match the currently selected element.

This tool is similar to the Inspect Element development tool that is found in most browsers. The **CSS Inspector** view allows you to see how the CSS rules are applied and the properties defined. Each rule that is displayed in this view includes a link to the line in the CSS file that defines the styles for the element that matches the rule. You can use the link to open the appropriate CSS file and edit the style rules. Once you've found the rule you want to edit, you can click the link in the top-right corner of that rule to open the CSS file in the editor.



There are two ways to open the CSS Inspector view:

1. Select **CSS Inspector** from the **Window > Show View** menu.
2. Select the **Inspect Styles** action from the contextual menu in **Author** mode.

Example Files Listings - The Simple Documentation Framework Files

This section lists the files used in the customization tutorials: the XML Schema, CSS files, XML files, XSLT stylesheets.

XML Schema files

sdf.xsd

This sample file can also be found in the *Oxygen SDK distribution* in the "oxygen sdk\samples\Simple Documentation Framework - SDF\framework\schema" directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.oxygenxml.com/sample/documentation"
  xmlns:doc="http://www.oxygenxml.com/sample/documentation"
  xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
  elementFormDefault="qualified">

  <xs:import
    namespace="http://www.oxygenxml.com/sample/documentation/abstracts"
    schemaLocation="abs.xsd"/>

  <xs:element name="book" type="doc:sectionType"/>
  <xs:element name="article" type="doc:sectionType"/>
  <xs:element name="section" type="doc:sectionType"/>

  <xs:complexType name="sectionType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element ref="abs:def" minOccurs="0"/>
      <xs:choice>
        <xs:sequence>
          <xs:element ref="doc:section"
            maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:choice maxOccurs="unbounded">
          <xs:element ref="doc:para"/>
        </xs:choice>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

```

```

        <xs:element ref="doc:ref"/>
        <xs:element ref="doc:image"/>
        <xs:element ref="doc:table"/>
    </xs:choice>
</xs:choice>
</xs:sequence>
</xs:complexType>

<xs:element name="para" type="doc:paragraphType"/>

<xs:complexType name="paragraphType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="b"/>
        <xs:element name="i"/>
        <xs:element name="link"/>
    </xs:choice>
</xs:complexType>

<xs:element name="ref">
    <xs:complexType>
        <xs:attribute name="location" type="xs:anyURI"
            use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="image">
    <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI"
            use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="table">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="customcol" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="width" type="xs:string"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="header">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td"
                            maxOccurs="unbounded"
                            type="doc:paragraphType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tr" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td"
                            type="doc:tdType"
                            maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:attribute name="width" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:complexType>

<xs:complexType name="tdType">
    <xs:complexContent>
        <xs:extension base="doc:paragraphType">
            <xs:attribute name="row_span"
                type="xs:integer"/>
            <xs:attribute name="column_span"
                type="xs:integer"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
</xs:schema>

```

abs.xsd

This sample file can also be found in the *Oxygen SDK distribution* in the "oxygensdk\samples\Simple Documentation Framework - SDF\framework\schema" directory.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
    "http://www.oxygenxml.com/sample/documentation/abstracts">
    <xs:element name="def" type="xs:string"/>
</xs:schema>

```

CSS Files

sdf.css

This sample file can also be found in the *Oxygen SDK distribution* in the oxygensdk\samples\Simple Documentation Framework - SDF\framework\css directory.

```

/* Element from another namespace */
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";

abs|def{
    font-family:monospace;
    font-size:smaller;
}
abs|def:before{
    content:"Definition:";
    color:gray;
}

/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
}

section{
    margin-left:1em;
    margin-top:1em;
}

section{
    -oxy-foldable:true;
    -oxy-not-foldable-child: title;
}

link[href]:before{
    display:inline;
    link:attr(href);
    content: "Click to open: " attr(href);
}

/* Title rendering*/
title{
    font-size: 2.4em;
    font-weight:bold;
}

* * title{
    font-size: 2.0em;
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}

book,
article{
    counter-reset:sect;
}
book > section,
article > section{
    counter-increment:sect;
}
book > section > title:before,
article > section > title:before{
    content: "Section: " counter(sect) " ";
}

/* Inlines rendering*/
b {
    font-weight:bold;
}

i {
    font-style:italic;
}

```

```

/*Table rendering */
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
    width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
    display:table-cell;
    border:1px solid navy;
    padding:1em;
}

image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}

```

XML Files

sdf_sample.xml

This sample file can also be found in the *Oxygen SDK distribution* in the "oxygensdk\samples\Simple Documentation Framework - SDF\framework" directory.

```

<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will explain
              different XML applications.</para>
    </section>
    <section>
        <title>Accessing XML data.</title>
        <section>
            <title>XSLT</title>
            <abs:def>Extensible stylesheet language
                  transformation (XSLT) is a language for
                  transforming XML documents into other XML
                  documents.</abs:def>
            <para>A list of XSL elements and what they do..</para>
            <table>
                <header>
                    <td>XSLT Elements</td>
                    <td>Description</td>
                </header>
                <tr>
                    <td>
                        <b>xsl:stylesheet</b>
                    </td>
                    <td>The <i>xsl:stylesheet</i> element is
                        always the top-level element of an
                        XSL stylesheet. The name
                        <i>xsl:transform</i> may be used
                        as a synonym.</td>
                    </tr>
                    <tr>
                        <td>
                            <b>xsl:template</b>
                        </td>
                        <td>The <i>xsl:template</i> element has
                            an optional mode attribute. If this
                            is present, the template will only
                            be matched when the same mode is
                            used in the invoking
                            <i>xsl:apply-templates</i>
                        </td>
                    </tr>
                </table>
            </section>
        </section>
    </book>

```

```

        element.</td>
    </tr>
    <tr>
        <td>
            <b>for-each</b>
        </td>
        <td>The xs:for-each element causes
            iteration over the nodes selected by
            a node-set expression.</td>
        </tr>
        <tr>
            <td column_span="2">End of the list</td>
        </tr>
    </table>
</section>
<section>
    <title>XPath</title>
    <abs:def>XPath (XML Path Language) is a terse
        (non-XML) syntax for addressing portions of
        an XML document. </abs:def>
    <para>Some of the XPath functions.</para>
    <table>
        <header>
            <td>Function</td>
            <td>Description</td>
        </header>
        <tr>
            <td>format-number</td>
            <td>The <i>format-number</i> function
                converts its first argument to a
                string using the format pattern
                string specified by the second
                argument and the decimal-format
                named by the third argument, or the
                default decimal-format, if there is
                no third argument</td>
        </tr>
        <tr>
            <td>current</td>
            <td>The <i>current</i> function returns
                a node-set that has the current node
                as its only member.</td>
        </tr>
        <tr>
            <td>generate-id</td>
            <td>The <i>generate-id</i> function
                returns a string that uniquely
                identifies the node in the argument
                node-set that is first in document
                order.</td>
        </tr>
    </table>
</section>
<section>
    <title>Documentation frameworks</title>
    <para>One of the most important documentation
        frameworks is DocBook.</para>
    <image
        href="http://www.xmlhack.com/images/docbook.png"/>
    <para>The other is the topic oriented DITA, promoted
        by OASIS.</para>
    <image
        href="http://www.oasis-open.org/images/standards/oasis_standard.jpg"
        />
</section>
</book>
```

XSL Files

sdf.xsl

This sample file can also be found in the *Oxygen SDK distribution* in the "oxygensdk\samples\Simple Documentation Framework - SDF\framework\xsl" directory.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xpath-default-namespace=
    "http://www.oxygenxml.com/sample/documentation">

    <xsl:template match="/">
        <html><xsl:apply-templates/></html>
    </xsl:template>

    <xsl:template match="section">
```

```

<xsl:apply-templates/>
</xsl:template>

<xsl:template match="image">
  
</xsl:template>

<xsl:template match="para">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="abs:def"
  xmlns:abs=
  "http://www.oxygenxml.com/sample/documentation/abstracts">
  <p>
    <u><xsl:apply-templates/></u>
  </p>
</xsl:template>

<xsl:template match="title">
  <h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="b">
  <b><xsl:apply-templates/></b>
</xsl:template>

<xsl:template match="i">
  <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="table">
  <table frame="box" border="1px">
    <xsl:apply-templates/>
  </table>
</xsl:template>

<xsl:template match="header">
  <tr>
    <xsl:apply-templates/>
  </tr>
</xsl:template>

<xsl:template match="tr">
  <tr>
    <xsl:apply-templates/>
  </tr>
</xsl:template>

<xsl:template match="td">
  <td>
    <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="header/header/td">
  <th>
    <xsl:apply-templates/>
  </th>
</xsl:template>

</xsl:stylesheet>

```

Author Component

The Author Component was designed as a separate product to provide the functionality of the standard **Author** mode. Recently (in version 14.2), the component API was extended to also allow multiple edit modes like **Text** and **Grid**. The component can be embedded either in a third-party standalone Java application or customized as a Java Web Applet to provide WYSIWYG-like XML editing directly in your web browser of choice.

The Author Component Startup Project for Java/Swing integrations is available online on the Oxygen XML Editor website as a Maven archetype. More information about the setup can be found [here](#)

Licensing

The licensing terms and conditions for the Author Component are defined in the [*<oXygen/> XML Editor SDK License Agreement*](#). To obtain the licensing terms and conditions and other licensing information as well, you can also contact our support team at support@oxygenxml.com. You may also obtain a free of charge evaluation license key for

development purposes. Any deployment of an application developed using the Author Component is also subject to the terms of the SDK agreement.

There are two main categories of Author Component integrations:

- 1. Integration for internal use.**

You develop an application which embeds the Author Component to be used internally (in your company or by you). You can buy and use Oxygen XML Editor standard licenses (either user-based or floating) to enable the Author Component in your application.

- 2. Integration for external use.**

Using the Author Component, you create an application that you distribute to other users outside your company (with a CMS for example). In this case you need to contact us to apply for a Value Added Reseller (VAR) partnership.

From a technical point of view, the Author Component provides the Java API to:

- Inject floating license server details in the Java code. The following link provides details about how to configure a floating license servlet: http://www.oxygenxml.com/license_server.html#floating_license_servlet.

```
AuthorComponentFactory.getInstance().init(frameworkZips, optionsZipURL, codeBase, appletID,
    //The servlet URL
    "http://www.host.com/servlet",
    //The HTTP credentials user name
    "userName",
    //The HTTP credentials password
    "password");
```

- Inject the licensing information key (for example the evaluation license key) directly in the component's Java code.

```
AuthorComponentFactory.getInstance().init(
    frameworkZips, optionsZipURL, codeBase, appletID,
    //The license key if it is a fixed license.
    licenseKey);
```

- Display the license registration dialog box. This is the default behavior in case a null license key is set using the API, this transfers the licensing responsibility to the end-user. The user can license an Author component using standard Oxygen XML Editor Editor/Author license keys. The license key will be saved to the local user's disk and on subsequent runs the user will not be asked anymore.

```
AuthorComponentFactory.getInstance().init(
    frameworkZips, optionsZipURL, codeBase, appletID,
    //Null license key, will ask the user.
    null);
```

Installation Requirements

Running the Author component as a Java applet requires:

- Oracle (Sun) Java JRE version 1.6 update 10 or newer;
- At least 100 MB disk space and 100MB free memory;
- The applet needs to be signed with a valid certificate and will request full access to the user machine, in order to store customization data (like options and framework files);
- A table of supported browsers can be found here:[Supported browsers and operating systems](#) on page 651.

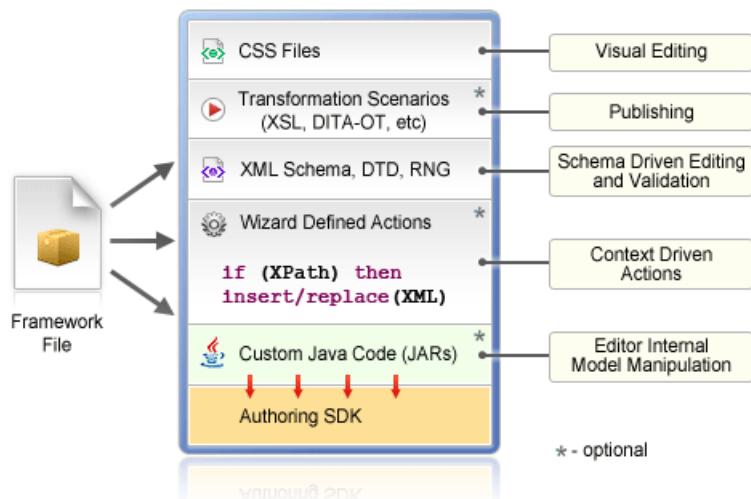
Running the Author component embedded in a third-party Java/Swing application requires:

- Oracle (Sun) Java JRE version 1.6 or newer;
- At least 100 MB disk space and 100MB free memory;

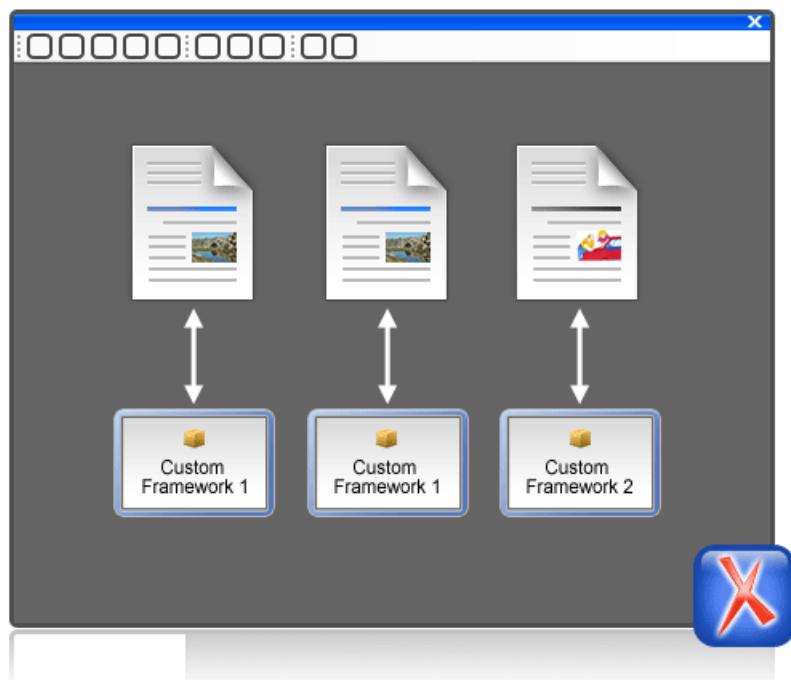
Customization

For a special type of XML, you can create a custom framework (which also works in an Oxygen standalone version). Oxygen XML Editor already has frameworks for editing DocBook, DITA, TEI, and so on. Their sources are available in [the Oxygen SDK](#). This custom framework is then packed in a zip archive and used to deploy the component.

The following diagram shows the components of a custom framework.



More than one framework can coexist in the same component and can be used at the same time for editing XML documents.



You can add on your custom toolbar all actions available in the standalone Oxygen XML Editor application for editing in the **Author** mode. You can also add custom actions defined in the framework customized for each XML type.

The Author component can also provide the *Outline*, *Model*, *Elements* and *Attributes* views which can be added to your own developed containers.

The main entry point for the Author Component Java API is the [AuthorComponentFactory](#) class.

Example - Customizing the DITA Framework

If you look inside the `bundle-frameworks\oxygen-frameworks` folder distributed with the [Author Component sample project](#), it contains a document type framework folder. Customizations which affect the framework/document type configuration for the component should first be done in an Oxygen standalone installation.

An Oxygen standalone installation comes with a `frameworks` folder which contains the `dita` framework located in `[OXYGEN_DIR]\frameworks\dita`. The `dita` framework contains a bundled **DITA-OT** distribution which contains the DTDs used for DITA editing. If your DTD specialization is a DITA OT plugin, it [should be installed](#) in the `[OXYGEN_DIR]\frameworks\dita\{DITA-OT}\plugins` folder.

To make changes to the DITA document type configuration, [open the Preferences dialog box](#) and go to **Document Type Association**. These changes will affect the `[OXYGEN_DIR]\frameworks\dita\dita.framework` configuration file.

After you do this you can re-pack the Author Component following the instructions from the `README.html` file located in the **oxygen-sample-applet** project. The Author Component Sample Project and the Oxygen standalone installation should be of the same version.

Packing a Fixed Set of Options

The Author Component shares a common internal architecture with the standalone application although it does not have **Preferences** dialog boxes. But the Author Component Applet can be configured to use a fixed set of user options on startup.

The sample project contains a module called `bundle-options`. The module contains a file called `options.xml` in the `oxygen-options` folder. Such an XML file can be obtained by exporting the options to an XML format from an installation of Oxygen XML Editor.

To create an *options file* in the Oxygen XML Editor:

- Make sure the options that you want to set are not [stored at project level](#).
- Set the values you want to impose as defaults in the [Preferences pages](#).
- Select **Options > Export Global Options**.

Deployment

The Author Component Java API allows you to use it in your Java application or as a Java applet. The JavaDoc for the API can be found [here](#). The sample project found in the `oxygen-sample-applet` module comes with Java sources (`ro.sync.ecss.samples/AuthorComponentSample.java`) demonstrating how the component is created, licensed and used in a Java application.

Web Deployment

The Author Component can be deployed as a Java Applet using the new Applet with JNLP Java technology, available in Oracle (Sun) Java JRE version 1.6 update 10 or newer.

The [sample project](#) demonstrates how the Author component can be distributed as an applet.

Here are the main steps you need to follow in order to deploy the Author component as a Java Applet:

- Follow the instructions [here](#) to setup the sample project and look for Java sources of the sample Applet implementation in the sample project **oxygen-sample-applet** module. They can be customized to fit your requirements.
- The `default.properties` configuration file must first be edited to specify your custom certificate information used to sign the applet libraries. You also have to specify the code base from where the applet will be downloaded.
- You can look inside the `web-resources/author-component-dita.html` and `web-resources/author-component-dita.js` sample Web resources to see how the applet is embedded in the page and how it can be controlled using JavaScript (to set and get XML content from it).
- The sample Applet `target/jnlp/author-component-dita.jnlp` file contains the list of used libraries. This list is automatically generated from the Maven dependencies of the project.

- The sample frameworks and options JAR archives can be found in the `bundle-frameworks` and `bundle-options` modules of the sample project.
- Use the Maven command `mvn package` to pack the component. More information are available [here](#). The resulting applet distribution is copied in the `target/jnlp/` directory. From this on, you can copy the applet files on your web server.

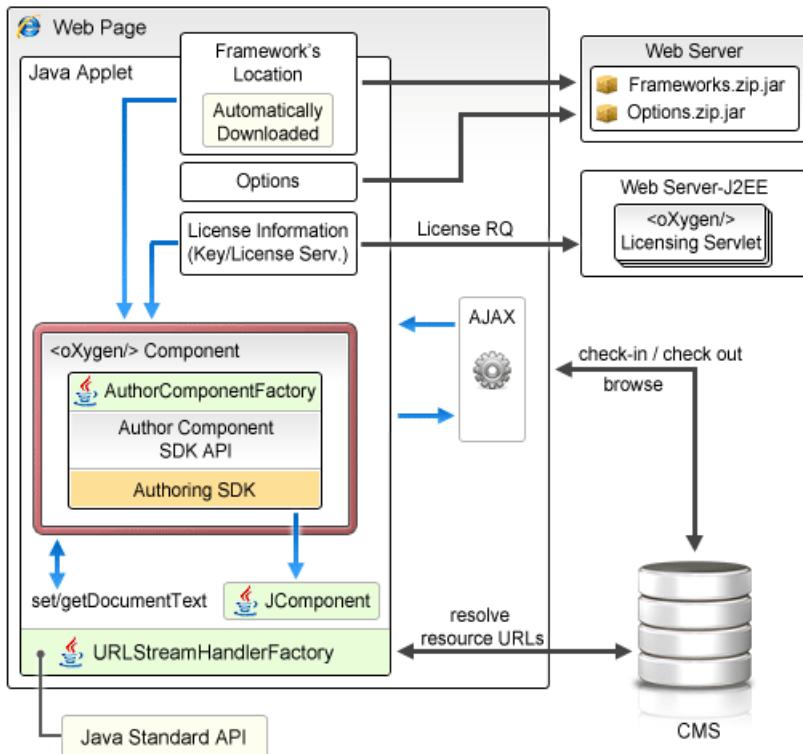


Figure 275: Oxygen XML Editor Author Component deployed as a Java applet

Generate a Testing Certificate For Signing an Applet

All jar files of an applet deployed on a remote Web server must be signed with the same certificate before the applet is deployed. The following steps describe how to generate a test certificate for signing the jar files. We will use the tool called `keytool` which is included in the Oracle's Java Development Kit.

- Create a `keystore` with a RSA encryption key.

Invoke the following in a command line terminal:

```
keytool -genkey -alias myAlias -keystore keystore.pkcs -storetype PKCS12
-keyalg RSA -keysize 2048 -dname "cn=your name here, ou=organization unit
name, o=organization name, c=US"
```

This command creates a `keystore` file called `keystore.pkcs`. The certificate attributes are specified in the `dname` parameter: common name of the certificate, organization unit name (for example *Purchasing* or *Sales Department*), organization name, country.

- Generate a self-signed certificate.

Invoke the following in a command line terminal:

```
keytool -selfcert -alias myAlias -keystore keystore.pkcs -storetype PKCS12
```

- Optionally display the certificate details in a human readable form.

First, the certificate must be exported to a separate file with the following command:

```
keytool -export -alias myAlias -keystore keystore.pcks -storetype PKCS12 -file certfile.cer
```

The certificate details are displayed with the command:

```
keytool -printcert -file certfile.cer
```

4. Edit the `default.properties` file and fill-in the parameters that hold the path to `keystore.pcks` file (`keystore` parameter), `keystore` type (`storetype` parameter, with JSK or PKCS12 as possible values), alias (`alias` parameter) and password (`password` parameter).
5. The jar files are automatically signed during the `package` phase of the Maven build.

Supported browsers and operating systems

The applet was tested for compatibility with the following browsers:

	IE 7	IE 8	IE 9	IE 10	IE 11	Firefox	Safari	Chrome	Opera
Vista	-	Passed	Passed	Passed	Passed	Passed	-	Passed	Passed
Windows 7	-	-	Passed	Passed	Passed	Passed	-	Passed	Passed
Windows 8	-	-	-	Passed	Passed	Passed	-	Passed	Passed
Mac OS X (10.6 - 10.9)	-	-	-	-	-	Passed	Passed	Failed	Passed
Linux Ubuntu 10	-	-	-	-	-	Passed	-	Failed	Passed

Communication between the Web Page and Java Applet

Applets can communicate with JavaScript code that runs in the Web Page. JavaScript code can call an applet Java methods and from the Java code you can invoke JavaScript code from the web page.

You are not limited to displaying only *Swing* dialog boxes from the applet. From the operations of an applet, you can invoke a JavaScript API that displays a web page and then obtains the data that has been filled in by the user.

Troubleshooting

When the applet fails to start:

1. Make sure that your web browser really runs the next generation Java plug-in and not the legacy Java plug-in.

For Windows and Mac OSX the procedure is straight forward. Some steps are given below for installing the Java plug-in on Linux.

Manual Installation and Registration of Java Plugin for Linux:

<http://www.oracle.com/technetwork/java/javase/manual-plugin-install-linux-136395.html>

2. Refresh the web page.
3. Remove the Java Webstart cache from the local drive and try again.
 - On Windows this folder is located in: %APPDATA%\LocalLow\Sun\Java\Deployment\cache;
 - On Mac OSX this folder is located in: /Users/user_name/Library/Caches/Java/cache;
 - On Linux this folder is located in: /home/user/.java/deployment/cache.
4. Remove the Author Applet Frameworks cache from the local drive and try again:

- On Windows Vista or 7 this folder is located in:
%APPDATA%\Roaming\com.oxygenxml.author.component;
 - On Windows XP this folder is located in: %APPDATA%\com.oxygenxml.author.component;
 - On Mac OSX this folder is located in:
/Users/user_name/Library/Preferences/com.oxygenxml.author.component;
 - On Linux this folder is located in: /home/user/.com.oxygenxml.author.component.
5. Problems sometimes occur after upgrading the web browser and/or the JavaTM runtime. Redeploy the applet on the server by running ANT in your Author Component project. However, doing this does not always fix the problem, which often lies in the web browser and/or in the Java plug-in itself.
6. Sometimes when the HTTP connection is slow on first time uses the JVM would simply shut down while the jars were being pushed to the local cache (i.e., first time uses). This shut down typically occurs while handling oxygen.jar. One of the reasons could be that some browsers (Firefox for example) implement some form of "Plugin hang detector" See
https://developer.mozilla.org/en/Plugins/Out_of_process_plugins/The_plugin_hang_detector.
7. If you are running the Applet using Safari on OS X and it has problems writing to disk or fails to start, do the following:
- in Safari, go to **Safari->Preferences->Security**;
 - select **Manage Website Settings**;
 - then select Java and for the **oxygenxml.com** entry choose the **Run in Unsafe mode** option.

Enable JavaWebstart logging on your computer to get additional debug information:

1. Open a console and run javaws -viewer;
2. In the **Advanced** tab, expand the **Debugging** category and select all boxes.
3. Expand the **Java console** category and choose **Show console**.
4. Save settings.
5. After running the applet, you will find the log files in:
 - On Windows this folder is located in: %APPDATA%\LocalLow\Sun\Java\Deployment\log;
 - On Mac OSX this folder is located in: /Users/user_name/Library/Caches/Java/log;
 - On Linux this folder is located in: /home/user/.java/deployment/log.

Avoiding Resource Caching

A Java plugin installed in a web browser caches access to all HTTP resources that the applet uses. This is useful in order to avoid downloading all the libraries each time the applet is run. However, this may have undesired side-effects when the applet presents resources loaded via HTTP. If such a resource is modified on the server and the browser window is refreshed, you might end-up with the old content of the resource presented in the applet.

To avoid such a behaviour, you need to edit the `ro.sync.ecss.samples.AuthorComponentSampleApplet` class and set a custom `URLStreamHandlerFactory` implementation. A sample usage is already available in the class, but it is commented-out for increased flexibility:

```
//THIS IS THE WAY IN WHICH YOU CAN REGISTER YOUR OWN PROTOCOL HANDLER TO THE JVM.
//THEN YOU CAN OPEN YOUR CUSTOM URLs IN THE APPLET AND THE APPLET WILL USE YOUR HANDLER
URL.setURLStreamHandlerFactory(new URLStreamHandlerFactory() {
    public URLStreamHandler createURLStreamHandler(String protocol) {
        if("http".equals(protocol) || "https".equals(protocol)) {
            return new URLStreamHandler() {
                @Override
                protected URLConnection openConnection(URL u) throws IOException {
                    URLConnection connection = new HttpURLConnection(u, null);
                    if(!u.toString().endsWith(".jar")) {
                        //Do not cache HTTP resources other than JARS
                        //By default the Java HTTP connection caches content for
                        //all URLs so if one URL is modified and then re-loaded in the
                        //applet the applet will show the old content.
                        connection.setDefaultUseCaches(false);
                    }
                    return connection;
                }
            };
        }
    }
});
```

```

    return null;
});
}

```

Adding MathML support in the Author Component Web Applet

By default the Author Component Web Applet project does not come with the libraries necessary for viewing and editing MathML equations in the Author page. You can view and edit MathML equations either by adding support for [JEuclid](#) or by adding support for [MathFlow](#).

Adding MathML support using JEuclid

By default, the `JEuclid` library is excluded from the `oxygen-sdk` artifact dependencies. To enable it, comment the following lines in the `pom.xml` file:

```

<exclusion>
  <artifactId>jeuclid-core</artifactId>
  <groupId>net.sourceforge.jeuclid</groupId>
</exclusion>

```

To edit specialized DITA Composite with MathML content, include the entire `[OXYGEN_DIR] / frameworks/mathml2` Mathml2 framework directory in the frameworks bundled with the component in the `bundle-frameworks` module. This directory is used to solve references to MathML DTDs.

Adding MathML support using MathFlow

In the `pom.xml` file add dependencies to the additional libraries used by the `MathFlow` library to parse MathML equations:

1. MFComposer.jar
2. MFExtraSymFonts.jar
3. MFSimpleEditor.jar
4. MFStructureEditor.jar
5. MFStyleEditor.jar

You can reference these additional libraries from the `MathFlow` SDK as in the example below:

```

<dependency>
  <groupId>com.dessci</groupId>
  <artifactId>MFComposer</artifactId>
  <version>1.0.0</version>
  <scope>system</scope>
  <systemPath>${MathFlowSDKDir}/lib/MFComposer.jar</systemPath>
</dependency>

```

In addition, you must obtain fixed `MathFlow` license keys for editing and composing `MathML` equations and register them using these API methods: `AuthorComponentFactory.setMathFlowFixedLicenseKeyForEditor` and `AuthorComponentFactory.setMathFlowFixedLicenseKeyForComposer`.

To edit specialized DITA Composite with `MathML` content, include the entire `[OXYGEN_DIR] / frameworks/mathml2` Mathml2 framework directory in the frameworks bundled with the component in the `bundle-frameworks` module. This directory is used to solve references to `MathML` DTDs.

More documentation is available on the [Design Science MathFlow](#) website.

Adding Support to Insert References from a WebDAV Repository

Already defined actions which insert references, like the **Insert Image Reference** action, display an URL chooser which allows you to select the **Browse Data Source Explorer** action. To use an already configured WebDAV connection in the Author Component, follow these steps:

1. Open a standalone Oxygen XML Editor 17.0 and configure a WebDAV connection;
2. Pack the *fixed set of options* from the standalone to use them with the Author Component Project;

3. In the Author Component, the defined connection still does not work when expanded because the additional JAR libraries used to browse the WebDAV repository are missing. By default, the `httpclient` dependency of the `oxygen-sdk` artifact is excluded. You can enable it by commenting the following lines:

```
<exclusion>
    <artifactId>httpclient</artifactId>
    <groupId>org.apache.httpcomponents</groupId>
</exclusion>
```

If you want to have a different WebDAV connection URL, user name and password depending on the user who has started the component, you have a more flexible approach using the API:

```
//DBConnectionInfo(String id, String driverName, String url, String user, String passwd, String host, String port)
DBConnectionInfo info = new DBConnectionInfo("WEBDAV", "WebDAV FTP", "http://host/webdav-user-root", "userName",
    "password", null, null);
AuthorComponentFactory.getInstance().setObjectProperty("database.stored.sessions1", new DBConnectionInfo[]
{info});
```

Using Plugins with the Author Component

To bundle Workspace Access plugins, that are developed for standalone application with the Author Component, follow these steps:

- The `bundle-plugins` module must contain the additional plugin directories in the `dropins` subdirectory. The content must also contain a `plugin.dtd` file.



Note:

Copy the `plugin.dtd` file from an `[OXYGEN_DIR]\plugins` folder.

- In the class which instantiates the `AuthorComponentFactory`, for example the `ro.sync.ecss.samples.AuthorComponentSample` class, call the methods `AuthorComponentFactory.getPluginToolbarCustomizers()`, `AuthorComponentFactory.getPluginViewCustomizers()` and `AuthorComponentFactory.getMenubarCustomizers()`, obtain the customizers which have been added by the plugins and call them to obtain the custom swing components that they contribute. There is a commented-out example for this in the `AuthorComponentSample.reconfigureActionsToolbar()` method for adding the toolbar from the **Acrolinx** plugin.



Important: As the Author Component is just a subset of the entire application, there is no guarantee that all the functionality of the plugin works.

Sample SharePoint Integration of the Author Component

This section presents the procedure to integrate the Author Component as a Java applet on a SharePoint site.

Author Component

The Author Component was designed as a separate product to provide the functionality of the standard **Author** mode. Recently (in version 14.2), the component API was extended to also allow multiple edit modes like **Text** and **Grid**. The component can be embedded either in a third-party standalone Java application or customized as a Java Web Applet to provide WYSIWYG-like XML editing directly in your web browser of choice.

The Author Component startup project for Java/Swing integrations is available online as a Maven archetype on the **<oXygen/> XML Editor** website. More information about the setup can be found [here](#)

Microsoft SharePoint®

Microsoft SharePoint® is a Web application platform developed by Microsoft®.

SharePoint comprises a multipurpose set of Web technologies backed by a common technical infrastructure. It provides the benefit of a central location for storing and collaborating on documents, which can significantly reduce emails and duplicated work in an organization. It is also capable of keeping track of the different versions created by different users.

Why Integrate the Author Component with SharePoint

The Author Component can be embedded in a SharePoint site as a Java applet. This is a simple and convenient way for you to retrieve, open, and save XML and XML related documents stored on your company's SharePoint server, directly from your web browser.

For example, let's say that you are working on a team project that uses the DITA framework for writing product documentation. You have the DITA Maps and topics stored on a SharePoint repository. By using a custom defined action from the contextual menu of a document, you can easily open it in the Author Component applet that is embedded in your SharePoint Documents page.

You can embed the applet either on a site that is located on a standalone SharePoint server, or on your company's Microsoft Office 365 account.

This example can be used as a starting point for other CMS integrations.

Integration Adjustments

Deploying Resources

You are able to embed the Author component in a SharePoint site as a Java Applet, using the new Applet with JNLP Java technology. Sign with a valid certificate the JNLP file and the associated JAR files that the applet needs.

Deploy these resources on a third party server (other than the SharePoint server). The Java applet downloads the resources as needed. If you deploy the JNLP and JAR files on the SharePoint server, the Java Runtime Environment will not be able to access the applet resources because it is not aware of the current authentication tokens from your browser. This causes the Java Class Loader to fail loading classes, making the applet unable to start.

Accessing Documents

One of the main challenges when integrating the Author Component applet in your SharePoint site is to avoid authenticating twice when opening a document resource stored in your SharePoint repository.

You have already signed in when you started the SharePoint session, but the applet is not aware of your current session. In this case every time the applet is accessing a document it will ask you to input your credentials again.

As a possible solution, do not execute HTTP requests directly from the Java code, but forward them to the web browser that hosts the applet, because it is aware of the current user session (authentication cookies).

To open documents stored on your SharePoint repository, register your own protocol handler to the JVM. We implemented a handler for both *http* and *https* protocols that forwards the HTTP requests to a JavaScript XMLHttpRequest object. This way, the browser that executes the JavaScript code is responsible for handling the authentication to the SharePoint site.

To install this handler, add the following line to your Java Applet code (in our case, in the `ro.sync.ecss.samples.AuthorComponentSampleApplet` class):

```
URL.setURLStreamHandlerFactory(new ro.sync.net.protocol.http.handlers.CustomURLStreamHandlerFactory(this));
```

To enable JavaScript calls from your Java applet code, set the MAYSCRIPT attribute to `true` in the `<applet>` element embedded in you HTML page:

```
<applet width="100%" height="600"
       code="ro.sync.ecss.samples.AuthorComponentSampleApplet"
       name="authorComponentAppName" id="authorComponentApplet"
       MAYSCRIPT="true">
    .....
</applet>
```

 **Tip:** In case the applet is not working, or you cannot open documents from your SharePoint repository, enable the debugging tools that come bundled with your Web Browser or the Java Console from your operating system to try to identify the cause of the problem.

Getting Started

To integrate the Author Component as a Java applet with your SharePoint site, you need the author component start-up project.

The project is available as a Maven archetype online. More information about the setup can be found [here](#).

An online demo applet is deployed at

<http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita-requirements.html>.

Customize Your Applet

Follow these steps to customize the Author Component Java applet:

1. Follow [this set of instructions](#) to setup the sample project and look for the Java sources (these can be customized to fit your requirements) of the sample applet implementation;



Note: The Java source files are located in the `src` folder of the `oxygen-sample-applet` module.

2. Look inside `web-resources/sharepoint/author-component-dita.aspx` and the associated `*.js` resources, to see how the applet is embedded in the page and how it can be controlled using JavaScript (to set and get XML content from it).

3. Edit the `default.properties` configuration to specify your custom certificate information, used to sign the applet libraries. Also, specify the code base from where the applet resources will be downloaded;

4. The sample Applet `target/jnlp/author-component-dita.jnlp` file contains the list of used libraries. This list is automatically generated from the Maven dependencies of the project. The sample frameworks and options JAR archives are located in the `bundle-frameworks` and `bundle-options` modules of the sample project..



Note: The JNLP file and the associated resources and libraries must be deployed on a non-SharePoint web server, otherwise the applet will not be loaded.

5. Use the Maven command `mvn package` to pack the component. More information are available [here](#). The resulting applet distribution is copied in the `target/jnlp/` directory. From now on, you can copy the applet files on your web server.

Add Resources to Your SharePoint Site

Copy the following resources to a sub-folder (in our example named `author-component`) of the `SitePages` folder from your SharePoint site, where you want to embed the applet:

1. **author-component-dita.aspx** - an HTML document containing the Java applet;



Note: It has an `.aspx` extension instead of `.html`. If you use the latter extension, the browser will download the HTML document instead of displaying it.



Note: Edit the `.aspx` file and change the value of the applet parameter `jnlp_href` to the URL of the deployed `author-component-dita.jnlp`. Keep in mind that the JNLP file should be deployed on a third party server. For example:

```
<applet>
    <param name="jnlp_href"
           value="http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita.jnlp" />
    .....
</applet>
```

2. **author-component-dita.css** - contains custom styling rules for the HTML document;

3. **author-component-dita.js** - contains JavaScript code, giving access to the Author Component contained by the Java applet;

4. **connectionUtil.js** - contains JavaScript utility methods.



Note: Replace the value of the `SPRootSiteURL` property with the URL of your SharePoint root site, without trailing `'/'`. This is used by the `openListItemInAuthor(itemUrl)` method, to compute the absolute URL of the list item that is to be opened in the Author applet.

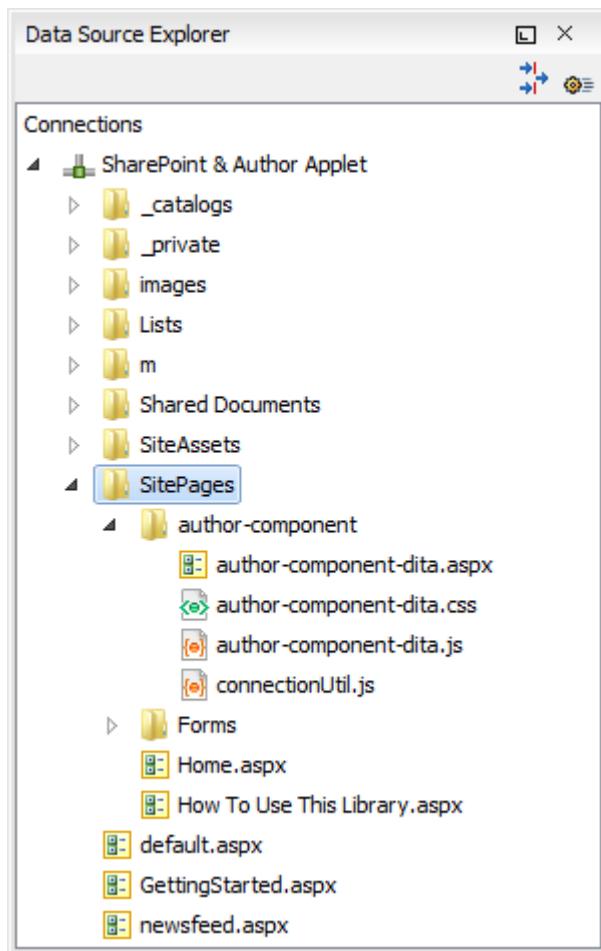
Copy Resources Using <oXygen/> XML Editor

You can use **<oXygen/> XML Editor** to copy your resources to the SharePoint server:

- Configure a new connection to your SharePoint site in the **Data Source Explorer** View.

 **Note:** To watch our video demonstration about connecting to repository located on a SharePoint server, go to http://www.oxygenxml.com/demo/SharePoint_Support.html.

- Browse your new SharePoint connection site and select the **SitePages** folder;
- Create a folder named **author-component** using the **New Folder** contextual menu action;
- Upload your resources to this folder using the **Import Files** contextual menu action.



Embed the Java Applet in Your SharePoint Site

To embed the Java Applet in your SharePoint site, edit the page that contains the applet and add a new Script Editor Web Part next to an existing Documents web part.

 **Note:** It is recommended that you deselect the **Enable Java content in the browser** option from the **Java Control Panel** until you finish editing the page. Otherwise, the browser will load the applet for every change that you will make.

Edit the page directly in your browser, following these steps:

- Navigate to the home page of your SharePoint site where you want to add the Author Component Java applet.
- Select the **Page** tab from the ribbon located at top of the page and click the **Edit** button.
- Select the **Insert** tab and click **Web Part**.
- In the **Categories** panel, select **Media and Content**.
- In the **Parts** panel, select the **Script Editor** Web Part.

6. Click the **Add** button to insert the selected Web Part to your page content.
7. Select the newly added Web Part.
8. Select the **Web Part** tab and click the **Web Part Properties** button.
9. Click the **Edit Snippet** link under your Web Part.
10. Insert the following HTML snippet to your newly created Web Part:

```
<div>
  <iframe
    id="appletIFrame"
    src="/applet/SitePages/author-component/author-component-dita.aspx"
    width="800px" height="850px">
  </iframe>
  <script type="text/JavaScript">
    function openInAuthor(itemUrl) {
      var appletFrame = document.getElementById("appletIFrame");
      var appletWin = appletFrame.contentWindow;
      appletWin.openListItemInAuthor(itemUrl);
    }
  </script>
</div>
```

The above HTML fragment contains an **IFrame** that points to the page where the Java applet resides. Replace the value of the **src** attribute with the path of the **author-component-dita.aspx** HTML page that you added earlier to the **SitePages** folder;

 **Note:** Use the **iframe** element from the HTML fragment with the expanded form (**<iframe></iframe>**). Otherwise, the Web Part will not display the target page of the frame.

11. Save the changes you made to the page.

 **Note:** Do not forget to select the **Enable Java content in the browser**, to allow the browser to load the Java applet.

Create a SharePoint Custom Action

To open a document from your SharePoint repository in the Author Component applet, add a new custom action to the contextual menu of your Documents Library:

1. Open your SharePoint site in **Microsoft SharePoint Designer®**;
2. Click **Lists and Libraries** in the **Navigation** pane;
3. Open the **Documents** library;
4. Go to the **Custom Actions** panel;
5. Click the **New** button to add a new custom action;
6. Give a name to the action, for example **Open In Oxygen XML Author**;
7. In the **Select the type of action** section, select the **Navigate to URL** option and enter the following text:

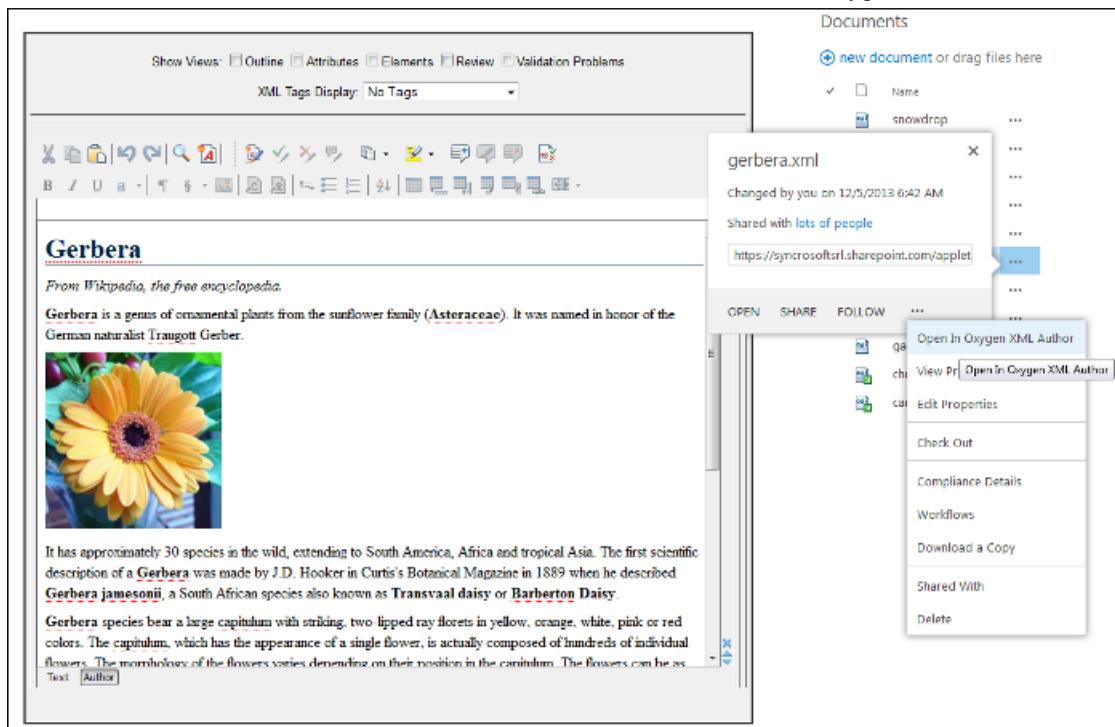
```
javascript:openInAuthor("{ItemUrl}")
```

 **Note:** This translates to a call to the `openInAuthor(itemUrl)` JavaScript function defined in the HTML fragment that was embedded in the Script Editor Web Part. The `{ItemUrl}` parameter will be expanded to the URL of the list item that the action is invoked on.

8. Click the **OK** button to save the action.

The Result

The Author Component applet embedded in a SharePoint site:



Frequently Asked Questions

Installation and Licensing

- What hosting options are available for applet delivery and licensing services (i.e., Apache, IIS, etc.)?

For applet delivery any web server. We currently use Apache to deploy the sample on our site. For the floating license server you would need a J2EE server, like Tomcat if you want to restrict the access to the licenses.

If you do not need the access restrictions that are possible with a J2EE server you can simplify the deployment of the floating license server by using the standalone version of this server. The standalone license server is a simple Java application that communicates with Author Component by TCP/IP connections.

- Are there any client requirements beyond the Java VM and (browser) Java Plug-In Technology?

Oracle (formerly Sun) Java JRE version 1.6 update 10 or newer. At least 200 MB disk space and 200MB free memory would be necessary for the Author Applet component.

- Are there any other client requirements or concerns that could make deployment troublesome (i.e., browser security settings, client-side firewalls and AV engines, etc.)?

The applet is signed and will request access to the user machine, in order to store customization data (frameworks). The applet needs to be signed by you with a valid certificate.

- How sensitive is the applet to the automatic Java VM updates, which are typically on by default (i.e., could automatic updates potentially "break" the run-time)?

The component should work well with newer Java versions but we cannot guarantee this.

- How and when are "project" related files deployed to the client (i.e., applet code, DTD, styling files, customizations, etc.)?

Framework files are downloaded on the first load of the applet. Subsequent loads will re-use the cached customization files and will be much faster.

6. For on-line demo (<http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita.html>), noted a significant wait during initial startup. Any other mechanisms to enhance startup time?

See the explanation above.

7. Does the XML Author component support multiple documents being open simultaneously? What are the licensing ramifications?

A single `AuthorComponentFactory` instance can create multiple `EditorComponentProvider` editors which can then be added and managed by the developer who is customizing the component in a Swing `JTabbedPane`. A single license (floating or user-based) is enough for this.

If you need to run multiple Java Applets or distinct Java processes using the Author component, the current floating license model allows for now only two concurrent components from the same computer when using the license servlet. An additional started component will take an extra license seat.

Another licensing technique would be to embed the license key in one of the jar libraries used by the applet. But you would need to implement your own way of determining how many users are editing using the Author applet.

8. Is there any internet traffic during an editing session (user actively working on the content, on the client side, in the XML Author component))?

No.

9. Does Oxygen XML Editor work in virtualized environments with terminal services, such as Citrix.

Oxygen XML Editor has been tested in virtualized environments with terminal services, including Citrix, and there have been no problems. We also have several customers that use Oxygen XML Editor in Citrix environments and we have not received any problem reports from them either.

For named licenses, you would normally have to deploy the license for each user or let each user be asked by Oxygen XML Editor for the license key and give it himself. Perhaps a better approach would be to create a file named "licensekey.txt"(without the quotes) and in this file paste the license key (all lines of text between the START-LICENSE-KEY and END-LICENSE-KEY markers). Copy this file to the Oxygen XML Editor installation folder. Any user who runs Oxygen XML Editor from this installation will use that license. Please note that this way of deploying the license key does not automatically limit the number of licenses to the specified number, so you should attempt to manually limit(keep evidence of) the number of users that are running Oxygen XML Editor to the number of licenses specified in the license key.

Functionality

1. How and when are saves performed back to the hosting server?

What you can see on our web site is just an example of the Author component (which is a Java Swing component) used in an Applet.

This applet is just for demonstration purposes. It's source can be at most a starting point for a customization. You should implement, sign and deploy your custom applet implementation.

The save operation could be implemented either in JavaScript by requesting the XML content from the Applet or in Java directly working with the Author component. You would be responsible to send the content back to the CMS.

2. Is there a particular XML document size (or range) when the Author applet would start to exhibit performance problems?

The applet has a total amount of used memory specified in the JNLP JavaWebstart configuration file which can be increased if necessary. By default it is 156 Mb. It should work comfortably with documents of 1-3 megabytes.

3. What graphic formats can be directly rendered in the XML Author component?

GIF, JPEG, PNG, BMP and SVG.

4. Can links be embedded to retrieve (from the server) and "play" other types of digital assets, such as audio or video files?

You could add listeners to intercept clicks and open the clicked links. This would require a good knowledge of the Oxygen SDK. The Author component can only render static images (no GIF animations).

5. Does the XML Author component provide methods for uploading ancillary files (new graphics, for instance) to the hosting server?

No.

6. Does the XML Author component provide any type of autosave functionality?

By default no but you could customize the applet that contains the author component to save its content periodically to a file on disk.

7. Assuming multiple documents can be edited simultaneously, can content be copied, cut and pasted from one XML Author component "instance" to another?

Yes.

8. Does the XML Author component support pasting content from external sources (such as a web page or a Microsoft Word document and, if so, to what extent?)

If no customizations are available the content is pasted as simple text. We provide customizations for the major frameworks (DITA, DocBook, TEI, etc) which use a conversion XSLT stylesheet to convert HTML content from clipboard to the target XML.

9. Can UTF-8 characters (such as Greeks, mathematical symbols, etc.) be inserted and rendered?

Any UTF-8 character can be inserted and rendered as long as the font used for editing supports rendering the characters. The font can be changed by the developers but not by the users. When using a logical font (which by default is *Serif* for the Author component) the JVM will know how to map all characters to glyphs. There is no character map available but you could implement one

Customization

1. Please describe, in very general terms, the menus, toolbars, context menu options, "helper panes", etc. that are available for the XML Author component "out of the box".

You can mount on your custom toolbar all actions available in the standalone Oxygen XML Editor application for editing in the Author page. This includes custom actions defined in the framework customized for each XML type.

The Author component also can provide the *Outline*, *Model*, *Elements* and *Attributes* views which can be added to your own panels (see sample applet).

2. Please describe, in general terms, the actions, project resources (e.g., DTD/Schema for validation purposes, CSS/XSL for styling, etc.) and typical level of effort that would be required to deploy a XML Author component solution for a customer with a proprietary DTD.

The Author internal engine uses CSS to render XML.

For a special type of XML you can create a custom framework (which also works in an Oxygen standalone version) which would also contain default schemas and custom actions. A simple framework would probably need 2-3 weeks development time. For a complex framework with many custom actions it could take a couple of months. Oxygen already has frameworks for editing DocBook, DITA, TEI, etc. Sources for them are available in [the Oxygen SDK](#).

More than one framework can coexist in the same Oxygen XML Editor instance (the desktop standalone version or the applet version) and can be used at the same time for editing XML documents.

3. Many customers desire a very simplistic interface for contributors (with little or no XML expertise) but a more robust XML editing environment for editors (or other users with more advanced XML expertise). How well does the XML Author component support varying degrees of user interface complexity and capability?

- *Showing/hiding menus, toolbars, helpers, etc.*

All the UI parts from the Author component are assembled by you. You could provide two applet implementations: one for advanced/power users and one for technical authors.

- *Forcing behaviors (i.e., ensuring change tracking is on and preventing it from being shut down)*

You could avoid placing the change tracking toolbar actions in the custom applet. You could also use API to turn change tracking ON when the content has been loaded.

- *Preventing access to "privileged" editor processes (i.e., accept/reject changes)*

You can remove the change tracking actions completely in a custom applet implementation. Including the ones from the contextual menu.

- *Presenting and/or describing XML constructs (i.e., tags) in "plain-English"*

Using our API you can customize what the Outline or Breadcrumb presents for each XML tag. You can also customize the in-place content completion list.

- *Presenting a small subset of the overall XML tag set (rather than the full tag set) for use by contributors (i.e., allowing an author to only insert Heading, Para and inline emphasis) Could varying "interfaces", with different mixes these capabilities and customizations, be developed and pushed to the user based on a "role" or a similar construct?*

The API allows for a content completion filter which also affects the *Elements* view.

4. Does the XML Author component API provide access to the XML document, for manipulation purposes, using common XML syntax such as DOM, XPath, etc.?

Yes, using the Author API.

5. Can custom dialog boxes be developed and launched to collect information in a "form" (with scripting behind to push tag the collection information and embed it in the XML document)?

Yes.

6. Can project resources, customizations, etc. be readily shared between the desktop and component versions of your XML Author product line?

A framework developed for the Desktop Oxygen application can then be bundled with an Author component in a custom applet. For example the Author demo applet from our web site is DITA-aware using the same framework as the Oxygen standalone distribution.

A custom version of the applet that includes one or more customized frameworks and user options can be built and deployed for non-technical authors by a technical savvy user using a built-in tool of Oxygen. All the authors that load the deployed applet from the same server location will share the same frameworks and options.

A custom editing solution can deploy one or more frameworks that can be used at the same time.

Creating and Running Automated Tests

If you have developed complex custom plugins and/or document types the best way to test your implementation and insure that further changes will not interfere with the current behavior is to make automated tests for your customization.

An Oxygen XML Editor installation standalone (Author or Editor) comes with a main `oxygen.jar` library located in the `[OXYGEN_DIR]`. That JAR library contains a base class for testing developer customizations named `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

Please see below some steps in order to develop JUnit tests for your customizations using the **Eclipse** workbench:

1. Create a new Eclipse Java project and copy to it the entire contents of the `[OXYGEN_DIR]`.
2. Add to the **Java Build Path->Libraries** tab all JAR libraries present in the `[OXYGEN_DIR]/lib` directory. Make sure that the main JAR library `oxygen.jar` or `oxygenAuthor.jar` is the first one in the Java classpath by moving it up in the **Order and Export** tab.
3. Click **Add Library** and add the JUnit libraries.

4. Create a new Java class which extends `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

5. Pass on to the constructor of the super class the following parameters:

- `File frameworksFolder` The file path to the frameworks directory. It can point to a custom frameworks directory where the custom framework resides.
- `File pluginsFolder` The file path to the plugins directory. It can point to a custom plugins directory where the custom plugins resides.
- `String licenseKey` The license key used to license the test class.

6. Create test methods which use the API in the base class to open XML files and perform different actions on them.

Your test class could look something like:

```
public class MyTestClass extends PluginWorkspaceTCBase {

    /**
     * Constructor.
     */
    public MyTestClass() throws Exception {
        super(new File("frameworks"), new File("plugins"),
              "-----START-LICENSE-KEY-----\n" +
              "\n" +
              "Registration_Name=Developer\n" +
              "\n" +
              "Company=\n" +
              "\n" +
              "Category=Enterprise\n" +
              "\n" +
              "Component=XML-Editor, XSLT-Debugger, Saxon-SA\n" +
              "\n" +
              "Version=14\n" +
              "\n" +
              "Number_of_Licenses=1\n" +
              "\n" +
              "Date=09-04-2012\n" +
              "\n" +
              "Trial=31\n" +
              "\n" +
              "SGN=MCwCFGNoEGJSeiC3XCYIyalvjzHhGhhqAhRNRDpEu8RIWb8icCJO7HqfVP4++A\|=|\|=|\n" +
              "\n" +
              "-----END-LICENSE-KEY-----");
    }

    /**
     * <p><b>Description:</b> TC for opening a file and using the bold operation</p>
     * <p><b>Bug ID:</b> EXM-20417</p>
     *
     * @author radu_coravu
     *
     * @throws Exception
     */
    public void testOpenFileAndBoldEXM_20417() throws Exception {
        WSEditor ed = open(new File("D:/projects/eXml/test/authorExtensions/dita/sampleSmall.xml").toURL());
        //Move caret
        moveCaretRelativeTo("Context", 1, false);

        //Insert <b>
        invokeAuthorExtensionActionForID("bold");
        assertEquals("<?xml version='1.0' encoding='utf-8'?>\n" +
                    "<!DOCTYPE task PUBLIC \"-//OASIS//DTD DITA Task//EN\""
                    + "\n\"http://docs.oasis-open.org/dita/v1.1/OS/dtd/task.dtd\">\n" +
                    "<task id=\"taskId\">\n" +
                    "  <title>Task <b>title</b></title>\n" +
                    "  <prolog/>\n" +
                    "  <taskbody>\n" +
                    "    <context>\n" +
                    "      <p>Context for the current task</p>\n" +
                    "    </context>\n" +
                    "    <steps>\n" +
                    "      <step>\n" +
                    "        <cmd>Task step.</cmd>\n" +
                    "      </step>\n" +
                    "    </steps>\n" +
                    "  </taskbody>\n" +
                    "</task>\n" +
                    "", getCurrentEditorXMLContent());
    }
}
```

API Frequently Asked Questions (API FAQ)

This section contains answers to common questions regarding the Oxygen XML Editor customisations using the *Oxygen SDK*, *Author Component*, or *Plugins*.

For additional questions, [contact us](#). The preferred approach is via email because API questions must be analysed thoroughly. We also provide code snippets in case they are required.

To stay up-to-date with the latest API changes, discuss issues and ask for solutions from other developers working with the Oxygen SDK, register to the [oxygen-SDK mailing list](#).

Difference Between a Document Type (Framework) and a Plugin Extension

Question

What is the difference between a Document Type (Framework) and a Plugin Extension?

Answer

Two ways of customising the application are possible:

1. Implementing a plugin.

A plugin serves a general purpose and influences any type of XML file that you open in Oxygen XML Editor.

For the Oxygen XML EditorPlugins API, Javadoc, samples, and documentation, go to
http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

2. Creating or modifying the document type which is associated to your specific XML vocabulary.

This document type is used to provide custom actions for your type of XML files and to mount them on the toolbar, menus, and contextual menus.

For example, if the application end users are editing DITA, all the toolbar actions which are specific for DITA are provided by the DITA Document Type. If you look in the Oxygen XML Editor Preferences->"Document Type Association" there is a "DITA" document type.

If you [edit that document type](#) in Oxygen XML Editor you will see that it has an Author tab in which it defines all custom DITA actions and adds them to the toolbars, main menus, contextual menus.

For information on developing your own document types, see [Authoring Customization Guide](#) on page 518.

If you look on disk in the:

```
[OXYGEN_DIR]\frameworks\dita
```

folder there is a file called `dita.framework`. That file gets updated when you edit a document type from the Oxygen XML Editor Preferences. Then you can share that updated file with all users.

The same folder contains some JAR libraries. These libraries contain custom Java operations which are called when the user presses certain toolbar actions.

We have an Oxygen SDK which contains the Java sources from all the DITA Java customizations:

http://www.oxygenxml.com/oxygen_sdk.html#XML_Editor_Authoring_SDK



Important: It is possible for a plugin to share the same classes with a framework. For further details, go to [How to Share the Classloader Between a Framework and a Plugin](#).

Dynamically Modify the Content Inserted by the Author

Question

Is there a way to insert typographic quotation marks instead of double quotes?

Answer

By using the API you can set a document filter to change the text that is inserted in the Author document. You can use this method to change the insertion of double quotes with the typographic quotes.

Here is some sample code:

```
authorAccess.getDocumentController().setDocumentFilter(new AuthorDocumentFilter() {
    /**
     * @see
     ro.sync.ecss.extensions.api.AuthorDocumentFilter#insertText(ro.sync.ecss.extensions.api.AuthorDocumentFilterBypass,
     int, java.lang.String)
     */
    @Override
    public void insertText(AuthorDocumentFilterBypass filterBypass, int offset, String toInsert) {
        if(toInsert.length() == 1 && "\"" .equals(toInsert)) {
            //User typed a quote but he actually needs a smart quote.
            //So we either have to add \u201E (start smart quote)
            //Or we add \u201C (end smart quote)
            //Depending on whether we already have a start smart quote inserted in the current paragraph.

            try {
                AuthorNode currentNode = authorAccess.getDocumentController().getNodeAtOffset(offset);
                int startofTextInCurrentNode = currentNode.getStartOffset();
                if(offset > startofTextInCurrentNode) {
                    Segment seg = new Segment();
                    authorAccess.getDocumentController().getChars(startofTextInCurrentNode, offset -
startofTextInCurrentNode, seg);
                    String previosTextInNode = seg.toString();
                    boolean insertStartQuote = true;
                    for (int i = previosTextInNode.length() - 1; i >= 0; i--) {
                        char ch = previosTextInNode.charAt(i);
                        if ('\u201C' == ch) {
                            //Found end of smart quote, so yes, we should insert a start one
                            break;
                        } else if ('\u201E' == ch) {
                            //Found start quote, so we should insert an end one.
                            insertStartQuote = false;
                            break;
                        }
                    }

                    if(insertStartQuote) {
                        toInsert = "\u201E";
                    } else {
                        toInsert = "\u201C";
                    }
                } catch (BadLocationException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("INSERT TEXT /" + toInsert + "/");
            super.insertText(filterBypass, offset, toInsert);
        }
    });
});
```

You can find the online Javadoc for AuthorDocumentFilter API here:

<http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro-sync/ecss/extensions/api/AuthorDocumentFilter.html>

An alternative to using a document filtering is the use of a

`ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandlerAdapter` which has clear callbacks indicating the source from where the API is called (Paste, Drag and Drop, Typing).

Split Paragraph on Enter (Instead of Showing Content Completion List)

Question

How to split the paragraph on Enter instead of showing the content completion list?

Answer

To obtain this behaviour, [edit your Document Type](#) and in the Author tab, Actions tab, add your own split action. This action must have the **Enter** shortcut key associated and must trigger your own custom operation which handles the split.

So, when you press **Enter**, your Java operation is invoked and it will be your responsibility to split the paragraph using the current API (probably creating a document fragment from the caret offset to the end of the paragraph, removing the content and then inserting the created fragment after the paragraph).

This solution has as a drawback. Oxygen XML Editor hides the content completion window when you press **Enter**. If you want to show allowed child elements at that certain offset, implement your own content proposals window using the `ro.sync.ecss.extensions.api.AuthorSchemaManager` API to use information from the associated schema.

Impose Custom Options for Authors**Question**

How to enable **Track Changes** at startup?

Answer

There are two ways to enable **Track Changes** for every document that you open:

1. You could [customise the default options](#) which are used by your authors and set the [*Track Changes Initial State option*](#) to Always On.
2. Use the API to toggle the Track Changes state after a document is opened in **Author** mode:

```
// Check the current state of Track Changes
boolean trackChangesOn = authorAccess.getReviewController().isTrackingChanges();
if (!trackChangesOn) {
    // Set Track Changes state to On
    authorAccess.getReviewController().toggleTrackChanges();
}
```

Highlight Content**Question**

How can we add custom highlights to the Author document content?

Answer

There are two types of highlights you can add:

1. Not Persistent Highlights. Such highlights are removed when the document is closed and then re-opened.

You can use the following API method:

```
ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPageBase.getHighlighter()
```

to obtain an [*AuthorHighlighter*](#) which allows you to add a highlight between certain offsets with a certain painter.

For example you can use this support to implement your custom spell checker.

2. Persistent Highlights. Such highlights are saved in the XML content as processing instructions.

You can use the following API method:

```
ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPageBase.getPersistentHighlighter()
```

to obtain an [*AuthorPersistentHighlighter*](#) which allows you to add a persistent highlight between certain offsets and containing certain custom properties and render it with a certain painter.

For example you can use this support to implement your own way of adding review comments.

How Do I Add My Custom Actions to the Contextual Menu?

The API methods `WSAuthorEditorPageBase.addPopUpMenuCustomizer` and `WSTextEditorPage.addPopUpMenuCustomizer` allow you to customize the contextual menu shown either in the Author or in the Text modes. The API is available both in the standalone application and in the Eclipse plugin.

Here's an elegant way to add from your Eclipse plugin extension actions to the Author page:

1. Create a pop-up menu customizer implementation:

```
import org.eclipse.jface.action.ContributionManager;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.menus.IMenuService;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.structure.AuthorPopupMenuCustomizer;
/**
 * This class is used to create the possibility to attach certain
 * menuContributions to the {@link ContributionManager}, which is used for the
 * popup menu in the Author Page of the Oxygen Editor.<br />
 * You just need to use the org.eclipse.ui.menus extension and add a
 * menuContribution with the locationURI: <b>menu:oxygen.authorpage</b>
 */
public class OxygenAuthorPagePopupMenuCustomizer implements
    AuthorPopupMenuCustomizer {

    @Override
    public void customizePopUpMenu(Object menuManagerObj,
        AuthorAccess authoraccess) {
        if (menuManagerObj instanceof ContributionManager) {
            ContributionManager contributionManager = (ContributionManager) menuManagerObj;
            IMenuService menuService = (IMenuService) PlatformUI.getWorkbench()
                .getActiveWorkbenchWindow().getService(IMenuService.class);

            menuService.populateContributionManager(contributionManager,
                "menu:oxygen.authorpage");
            contributionManager.update(true);
        }
    }
}
```

2. Add a workbench listener and add the pop-up customizer when an editor is opened in the Author page:

```
Workbench.getInstance().getActiveWorkbenchWindow().getPartService().addPartListener(
    new IPartListener() {
        @Override
        public void partOpened(IWorkbenchPart part) {
            if(part instanceof ro.sync.ecml.workspace.api.editor.WSEditor) {
                WSEditorPage currentPage = ((WSEditor)part).getCurrentPage();
                if(currentPage instanceof WSAuthorEditorPage) {
                    ((WSAuthorEditorPage)currentPage).addPopUpMenuCustomizer(new OxygenAuthorPagePopupMenuCustomizer());
                }
            }
        }
    });
....
```

3. Implement the extension point in your `plugin.xml`:

```
<extension
    point="org.eclipse.ui.menus">
    <menuContribution
        allPopups="false"
        locationURI="menu:oxygen.authorpage">
        <command
            commandId="eu.doccenter.kgu.client.tagging.removeTaggingFromOxygen"
            style="push">
        </command>
    </menuContribution>
</extension>
```

Adding Custom Callouts

Question

I'd like to highlight validation errors, instead of underlining them, for example changing the text background color to light red (or yellow). Also I like to let oxygen write a note about the error type into the author view directly at the error position, like "[value "text" not allowed for attribute "type"]". Is this possible using the API?

Answer

The Plugins API allows setting a `ValidationProblemsFilter` which gets notified when automatic validation errors are available. Then you can map each of the problems to an offset range in the Author page using the API `WSTextBasedEditorPage.getStartEndOffsets(DocumentPositionedInfo)`. For each of those offsets you can add either persistent or non-persistent highlights. If you add persistent highlights you can also customize callouts to appear for each of them, the downside is that they need to be removed before the document gets saved. The end result would look something like:

Keywords:

- z
- hard drive
- configure

Context:

First check the documentation that came with your storage device. If the device requires configuring, follow the steps below.

Step 1

Step 2
Otherwise, your drive should come with software. Use this software to format and partition your drive.

Step 3
Once your drive is configured, restart the system. Just for fun. But be sure to remove any vendor software from your system before doing so.

Problem The content of element type "step" is incomplete, it must match "((note|hazardstatement)*, cmd, (choices|choicetable|info|itemgroup|stepxmp|substeps|tutorialinfo)*, stepresult)".

Problem Attribute "a" must be declared for element type "step".

Here is a small working example:

```
/**
 * Plugin extension - workspace access extension.
 */
public class CustomWorkspaceAccessPluginExtension
    implements WorkspaceAccessPluginExtension {

    /**
     * @see ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension
     #applicationStarted(ro.sync.exml.workspace.api.standalone.StandalonePluginWorkspace)
     */
    public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
        pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
            /**
             * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
             */
            @Override
            public void editorOpened(URL editorLocation) {
                final WSEditor currentEditor = pluginWorkspaceAccess.getEditorAccess(editorLocation,
                    StandalonePluginWorkspace.MAIN_EDITING_AREA);
                WSEditorPage currentPage = currentEditor.getCurrentPage();
                if(currentPage instanceof WSAuthorEditorPage) {
                    final WSAuthorEditorPage currentAuthorPage = (WSAuthorEditorPage)currentPage;
                    currentAuthorPage.getPersistentHighlighter().setHighlightRenderer(new PersistentHighlightRenderer())
                }
                @Override
                public String getTooltip(AuthorPersistentHighlight highlight) {
                    return highlight.getClonedProperties().get("message");
                }
                @Override
                public HighlightPainter getHighlightPainter(AuthorPersistentHighlight highlight) {
                    //Depending on severity could have different color.
                }
            }
        });
    }
}
```

```

        ColorHighlightPainter painter = new ColorHighlightPainter(Color.COLOR_RED, -1, -1);
        painter.setBgColor(Color.COLOR_RED);
        return painter;
    }
});
currentAuthorPage.getReviewController()
    .getAuthorCalloutsController().setCalloutsRenderingInformationProvider(
        new CalloutsRenderingInformationProvider() {
    @Override
    public boolean shouldRenderAsCallout(AuthorPersistentHighlight highlight) {
        //All custom highlights are ours
        return true;
    }
    @Override
    public AuthorCalloutRenderingInformation getCalloutRenderingInformation(
        final AuthorPersistentHighlight highlight) {
        return new AuthorCalloutRenderingInformation() {
            @Override
            public long getTimestamp() {
                //Not interesting
                return -1;
            }
            @Override
            public String getContentFromTarget(int limit) {
                return "";
            }
            @Override
            public String getComment(int limit) {
                return highlight.getClonedProperties().get("message");
            }
            @Override
            public Color getColor() {
                return Color.COLOR_RED;
            }
            @Override
            public String getCalloutType() {
                return "Problem";
            }
            @Override
            public String getAuthor() {
                return "";
            }
            @Override
            public Map<String, String> getAdditionalData() {
                return null;
            }
        };
    }
);
currentEditor.addValidationProblemsFilter(new ValidationProblemsFilter() {
    List<int[]> lastStartEndOffsets = new ArrayList<int[]>();
    /**
     * @see ro.sync.exml.workspace.api.editor.validation.ValidationProblemsFilter
     #filterValidationProblems(ro.sync.exml.workspace.api.editor.validation.ValidationProblems)
     */
    @Override
    public void filterValidationProblems(ValidationProblems validationProblems) {
        List<int[]> startEndOffsets = new ArrayList<int[]>();
        List<DocumentPositionedInfo> problemsList = validationProblems.getProblemsList();
        if(problemsList != null) {
            for (int i = 0; i < problemsList.size(); i++) {
                try {
                    startEndOffsets.add(currentAuthorPage.getStartEndOffsets(problemsList.get(i)));
                } catch (BadLocationException e) {
                    e.printStackTrace();
                }
            }
            if(lastStartEndOffsets.size() != startEndOffsets.size()) {
                //Continue
            } else {
                boolean equal = true;
                for (int i = 0; i < startEndOffsets.size(); i++) {
                    int[] o1 = startEndOffsets.get(i);
                    int[] o2 = lastStartEndOffsets.get(i);
                    if(o1 == null && o2 == null) {
                        //Continue
                    } else if(o1 != null && o2 != null
                        && o1[0] == o2[0] && o1[1] == o2[1]){
                        //Continue
                    } else {
                        equal = false;
                        break;
                    }
                }
                if(equal) {
                    //Same list of problems already displayed.
                    return;
                }
            }
        }
    }
});

```

```

        }
    }
    //Keep last used offsets.
    lastStartEndOffsets = startEndOffsets;
    try {
        if(! SwingUtilities.isEventDispatchThread()) {
            SwingUtilities.invokeAndWait(new Runnable() {
                @Override
                public void run() {
                    //First remove all custom highlights.
                    currentAuthorPage.getPersistentHighlighter().removeAllHighlights();
                }
            });
        }
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    } catch (InvocationTargetException e1) {
        e1.printStackTrace();
    }
    if(problemsList != null) {
        for (int i = 0; i < problemsList.size(); i++) {
            //A reported problem (could be warning, could be error).
            DocumentPositionedInfo dpi = problemsList.get(i);
            try {
                final int[] currentOffsets = startEndOffsets.get(i);
                if(currentOffsets != null) {
                    //These are offsets in the Author content.
                    final LinkedHashMap<String, String> highlightProps = new LinkedHashMap<String,
String>();
                    highlightProps.put("message", dpi.getMessage());
                    highlightProps.put("severity", dpi.getSeverityAsString());
                    if(! SwingUtilities.isEventDispatchThread()) {
                        SwingUtilities.invokeAndWait(new Runnable() {
                            @Override
                            public void run() {
                                currentAuthorPage.getPersistentHighlighter().addHighlight(
                                    currentOffsets[0], currentOffsets[1] - 1, highlightProps);
                            }
                        });
                    }
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
        }
    }
    currentEditor.addEditorListener(new WSEditorListener() {
        /**
         * @see ro.sync.exml.workspace.api.listeners.WSEditorListener#editorAboutToBeSavedVeto(int)
         */
        @Override
        public boolean editorAboutToBeSavedVeto(int operationType) {
            try {
                if(! SwingUtilities.isEventDispatchThread()) {
                    SwingUtilities.invokeAndWait(new Runnable() {
                        @Override
                        public void run() {
                            //Remove all persistent highlights before saving
                            currentAuthorPage.getPersistentHighlighter().removeAllHighlights();
                        }
                    });
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
            return true;
        }
    });
}
}, StandalonePluginWorkspace.MAIN_EDITING_AREA);

/**
 * @see ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension#applicationClosing()
 */
public boolean applicationClosing() {
    return true;
}
}

```

Change the DOCTYPE of an Opened XML Document

Question

How to change the DOCTYPE of a document opened in the **Author** mode?

Answer

The following API:

```
ro.sync.ecss.extensions.api.AuthorDocumentController.getDoctype()
```

allows you to get the DOCTYPE of the current XML file opened in the Author page.

There is also an API method available which would allow you to set the DOCTYPE back to the XML:

```
ro.sync.ecss.extensions.api.AuthorDocumentController.setDoctype(AuthorDocumentType)
```

Here is an example of how this solution would work:

```
AuthorDocumentType dt = new AuthorDocumentType("article", "testSystemID", "testPublicID",
    "<!DOCTYPE article PUBLIC \"testPublicID\" \"testSystemID\">");
docController.setDoctype(dt);
```

Basically you could take the entire content from the existing DOCTYPE,

```
ro.sync.ecss.extensions.api.AuthorDocumentType.getContent()
```

modify it to your needs, and create another `AuthorDocumentType` object with the new content and with the same public, system IDs.

For example you could use this API if you want to add unparsed entities in the XML DOCTYPE.

Customize the Default Application Icons for Toolbars/Menus

Question

How can we change the default icons used for the application built-in actions?

Answer

If you look inside the main JAR library `[OXYGEN_DIR]\lib\oxygen.jar` or `[OXYGEN_DIR]\lib\author.jar` it contains an `images` folder in which all the images which we use for our buttons, menus, and toolbars exist.

In order to overwrite them with your own creations:

1. In the `[OXYGEN_DIR]\lib` directory create a folder called `endorsed`;
2. In the `endorsed` folder create another folder called `images`;
3. Add your own images in the `images` folder.

You can use this mechanism to overwrite any kind of resource located in the main Oxygen JAR library. The folder structure in the `endorsed` directory and in the main Oxygen JAR must be identical.

Disable Context-Sensitive Menu Items for Custom Author Actions

Question

Is there a way to disable menu items for custom Author actions depending on the cursor context?

Answer

By default Oxygen XML Editor does not toggle the enabled/disabled states for actions based on whether the activation XPath expressions for that certain Author action are fulfilled. This is done because the actions can be many and evaluating XPath expression on each caret move can lead to performance problems. But if you have your own `ro.sync.ecss.extensions.api.ExtensionsBundle` implementation you can overwrite the method:

```
ro.sync.ecss.extensions.api.ExtensionsBundle.createAuthorExtensionStateListener()
```

and when the extension state listener gets activated you can use the API like:

```
/**  
 * @see  
 ro.sync.ecss.extensions.api.AuthorExtensionStateListener#activated(ro.sync.ecss.extensions.api.AuthorAccess)  
 */  
public void activated(final AuthorAccess authorAccess) {  
  
    //Add a caret listener to enable/disable extension actions:  
    authorAccess.getEditorAccess().addAuthorCaretListener(new AuthorCaretListener() {  
        @Override  
        public void caretMoved(AuthorCaretEvent caretEvent) {  
            try {  
                Map<String, Object> authorExtensionActions =  
                    authorAccess.getEditorAccess().getActionsProvider().getAuthorExtensionActions();  
                //Get the action used to insert a paragraph. It's ID is "paragraph"  
                AbstractAction insertParagraph = (AbstractAction) authorExtensionActions.get("paragraph");  
                //Evaluate an XPath expression in the context of the current node in which the caret is located  
                Object[] evaluateXPath = authorAccess.getDocumentController().evaluateXPath(".[ancestor-or-self::p]",  
                    false, false, false, false);  
                if(evaluateXPath != null && evaluateXPath.length > 0 && evaluateXPath[0] != null) {  
                    //We are inside a paragraph, disable the action.  
                    insertParagraph.setEnabled(false);  
                } else {  
                    //Enable the action  
                    insertParagraph.setEnabled(true);  
                }  
            } catch (AuthorOperationException e) {  
                e.printStackTrace();  
            }  
        }  
    });  
}
```

When the extension is deactivated you should remove the caret listener in order to avoid adding multiple caret listeners which perform the same functionality.

Dynamic Open File in Oxygen XML Editor Distributed via JavaWebStart

Question

How can we dynamically open a file in an Oxygen XML Editor distributed via JWS?

Answer

The JWS packager ANT build file which comes with Oxygen XML Editor signs by default the JNLP file (this means that a copy of it is included in the main JAR library) in this step:

```
<copy file="${outputDir}/${packageName}/${productName}.jnlp" tofile="${home}/JNLP-INF/APPLICATION.JNLP"/>
```

Sigining the JNLP file is required by newer Java versions and means that it is impossible to automatically generate a JNLP file containing some dynamic arguments. The solution is to use the signed JNLP template feature of Java 7, bundle inside the JAR library a signed APPLICATION_TEMPLATE.JNLP instead of an APPLICATION.JNLP with a wildcard command line argument:

```
<application-desc main-class="ro.sync.jws.JwsDeployer">  
    <argument>*</argument>  
</application-desc>
```

Then you can replace the wildcard in the external placed JNLP to the actual, dynamic command line arguments value.

A different approach (more complicated though) would be to have the JNLP file signed and always referenced as a URL argument a location like this:

```
http://path/to/server/redirectEditedURL.php
```

When the URL gets clicked on the client side you would also call a PHP script on the server side which would update the redirect location for `redirectEditedURL.php` to point to the clicked XML resource. Then the opened Oxygen XML Editor would try to connect to the redirect PHP and be redirected to open the XML.

Change the Default Track Changes (Review) Author Name

Question

How can we change the default author name used for Track Changes in the Author Component?

Answer

The Track Changes (Review) Author name is determined in the following order:

1. **API** - The review user name can be imposed through the following API:

```
ro.sync.ecss.extensions.api.AuthorReviewController.setReviewerAuthorName(String)
```

2. **Options** - If the author name was not imposed from the API, it is determined from the **Author** option set from the *Review preferences page*.
3. **System properties** - If the author name was not imposed from the API or from the application options then the following system property is used:

```
System.getProperty("user.name")
```

So, to impose the Track Changes author, use one of the following approaches:

1. Use the API to impose the reviewer Author name. Here is the online Javadoc of this method:
[http://www.oxygenxml.com/InstDataEditor/SDK/javadoc/ro-sync/ecss/extensions/api/AuthorReviewController.html#setReviewerAuthorName\(java.lang.String\)](http://www.oxygenxml.com/InstDataEditor/SDK/javadoc/ro-sync/ecss/extensions/api/AuthorReviewController.html#setReviewerAuthorName(java.lang.String))
2. *Customise the default options* and set a specific value for the reviewer Author name option.
3. Set the value of `user.name` system property when the applet is initialising and before any document is loaded.

Multiple Rendering Modes for the Same Author Document

Question

How can we add multiple buttons, each showing different visualisation mode of the same Author document (by associating additional/different CSS style sheet)?

Answer

In the toolbar of the **Author** mode there is a **Styles** drop-down list that contains alternative CSS styles for the same document. To add an alternative CSS stylesheet, *open the Preferences dialog box*, go to **Document Type Association**, select the document type associated with your documents and press **Edit**. In the **Document Type** dialog box that appears, go to the **Author** tab, and in the **CSS** subtab add references to alternate CSS stylesheets.

For example, one of the alternate CSS stylesheets that we offer for the DITA document type is located here:

```
[OXYGEN_DIR]/frameworks/dita/css_classed/hideColspec.css
```

If you open it, you will see that it imports the main CSS and then adds selectors of its own.

Obtain a DOM Element from an `AuthorNode` or `AuthorElement`

Question

Can a DOM Element be obtained from an `AuthorNode` or an `AuthorElement`?

Answer

No, a DOM Element cannot be obtained from an `AuthorNode` or an `AuthorElement`. The `AuthorNode` structure is also hierarchical but the difference is that all the text content is kept in a single text buffer instead of having individual text nodes.

We have an image in the Javadoc which explains the

situation:<http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/node/AuthorDocumentFragment.html>

Print Document Within the Author Component

Question

Can a document be printed within the Author Component?

Answer

You can use the following API method to either print the Author document content to the printer or to show the Print Preview dialog box, depending on the `preview` parameter value:

```
AuthorComponentProvider.print(boolean preview)
```

Here is the online Javadoc for this method:

[http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/component/AuthorComponentProvider.html#print\(boolean\)](http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/component/AuthorComponentProvider.html#print(boolean))

Running XSLT or XQuery Transformations

Question

Can I run XSL 2.0 / 3.0 transformation with Saxon EE using the oXygen SDK?

Answer

The API class `ro.sync.exml.workspace.api.util.XMLUtilAccess` allows you to create an XSLT Transformer which implements the JAXP interface `javax.xml.transform.Transformer`. Then this type of transformer can be used to transform XML. Here's just an example of transforming when you have an `AuthorAccess` API available:

```
InputSource is = new org.xml.sax.InputSource(URLUtil.correct(new File("test/personal.xsl")).toString());
xslSrc = new SAXSource(is);
javax.xml.transform.Transformer transformer = authorAccess.getXMLUtilAccess().createXSLTTransformer(xslSrc,
null, AuthorXMLUtilAccess.TRANSFORMER_SAXON_ENTERPRISE_EDITION);
transformer.transform(new StreamSource(new File("test/personal.xml")), new StreamResult(new
File("test/personal.html")));
```

If you want to create the transformer from the plugins side, you can use this method instead:

```
ro.sync.exml.workspace.api.PluginWorkspace.getXMLUtilAccess().
```

Use Different Rendering Styles for Entity References, Comments or Processing Instructions

Question

Is there a way to display entity references in the **Author** mode without the distinct gray background and tag markers?

Answer

There is a built-in CSS stylesheet in the Oxygen XML Editor libraries which is used when styling content in the **Author** mode, no matter what CSS you use. This CSS has the following content:

```

@namespace oxy url('http://www.oxygenxml.com/extensions/author');
@namespace xi "http://www.w3.org/2001/XInclude";
@namespace xlink "http://www.w3.org/1999/xlink";
@namespace svg "http://www.w3.org/2000/svg";
@namespace mml "http://www.w3.org/1998/Math/MathML";

oxy|document {
    display:block !important;
}

oxy|cdata {
    display:morph !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|processing-instruction {
    display:block !important;
    color: rgb(139, 38, 201) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|comment {
    display:morph !important;
    color: rgb(0, 100, 0) !important;
    background-color:rgb(255, 255, 210) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|reference:before,
oxy|entity[href]:before{
    link: attr(href) !important;
    text-decoration: underline !important;
    color: navy !important;

    margin: 2px !important;
    padding: 0px !important;
}

oxy|reference:before {
    display: morph !important;
    content: url(..../images/editContent.gif) !important;
}

oxy|entity[href]:before{
    display: morph !important;
    content: url(..../images/editContent.gif) !important;
}

oxy|reference,
oxy|entity {
    editable:false !important;
    background-color: rgb(240, 240, 240) !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|reference {
    display:morph !important;
}

oxy|entity {
    display:morph !important;
}

oxy|entity[href] {
    border: 1px solid rgb(175, 175, 175) !important;
    padding: 0.2em !important;
}

xi|include {
    display:block !important;
    margin-bottom: 0.5em !important;
    padding: 2px !important;
}

```

```

xi|include:before,
xi|include:after{
    display:inline !important;
    background-color:inherit !important;
    color:#444444 !important;
    font-weight:bold !important;
}

xi|include:before {
    content:url(..../images/link.gif) attr(href) !important;
    link: attr(href) !important;
}
xi|include[xpointer]:before {
    content:url(..../images/link.gif) attr(href) " " attr(xpointer) !important;
    link: oxy_concat(attr(href), "#", attr(xpointer)) !important;
}

xi|fallback {
    display:morph !important;
    margin: 2px !important;
    border: 1px solid #CB0039 !important;
}

xi|fallback:before {
    display:morph !important;
    content:"XInclude fallback: " !important;
    color:#CB0039 !important;
}

oxy|doctype {
    display:block !important;
    background-color: transparent !important;
    color:blue !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 2px !important;
}

oxy|error {
    display:morph !important;
    editable:false !important;
    white-space:pre !important;
    color: rgb(178, 0, 0) !important;
    font-weight:bold !important;
}

*[xlink|href]:before {
    content:url(..../images/link.gif);
    link: attr(xlink|href) !important;
}

/*No direct display of the MathML and SVG images.*/
svg|svg{
    display:inline !important;
    white-space: trim-when-ws-only;
}
svg|svg svg|*{
    display:none !important;
    white-space: normal;
}

mml|math{
    display:inline !important;
    white-space: trim-when-ws-only;
}
mml|math mml|*{
    display:none !important;
    white-space: normal;
}

```

In the CSS used for rendering the XML in **Author** mode do the following:

- import the special Author namespace;
- use a special selector to customize the entity node.

Example:

```

@namespace oxy url('http://www.oxygenxml.com/extensions/author');
oxy|entity {
    background-color: inherit !important;
    margin:0px !important;
    padding: 0px !important;
    -oxy-display-tags:none;
}

```

You can overwrite styles in the predefined CSS in order to custom style comments, processing instructions and *CData* sections. You can also customize the way in which `xi:include` elements are rendered.

Insert an Element with all the Required Content

Question

I'm inserting a DITA *image* XML element, using the Author API, which points to a certain resource and has required content. Can the required content be automatically inserted by the application?

Answer

The API `ro.sync.ecss.extensions.api.AuthorSchemaManager` can propose valid elements which can be inserted at the specific offset. Using the method

`AuthorSchemaManager.createAuthorDocumentFragment(CIElement)` you can convert the proposed elements to document fragments (which have all the required content filled in) which can then be inserted in the document.

Obtain the Current Selected Element Using the Author API

Question

If in the **Author** mode, an element is fully selected, I would like to perform an action on it. If not, I would like to perform an action on the node which is located at the caret position. Is this possible via the API?

Answer

When an element is fully selected by the user the selection start and end offsets are actually outside of the node's offset bounds. So using `AuthorDocumentController.getNodeAtOffset` will actually return the parent of the selected node. We have some special API which makes it easier for you to determine this situation:

WSAuthorEditorPageBase.getFullySelectedNode().

```
AuthorDocumentController controller = authorPageAccess.getDocumentController();
AuthorAccess authorAccess = authorPageAccess.getAuthorAccess();
int caretOffset = authorAccess.getEditorAccess().getCaretOffset();
```

```

AuthorElement nodeAtCaret = (AuthorElement) authorAccess.getEditorAccess().getFullySelectedNode();
if (nodeAtCaret == null) {
    //We have no fully selected node. We can look at the caret offset.
    nodeAtCaret = (AuthorElement) authorAccess.getDocumentController().getNodeAtOffset(caretOffset);
    //Or we could look at the selection start and end, see which node is the parent of each offset and get the
    closest common ancestor.
}

```

Debugging a Plugin Using the Eclipse Workbench

To debug problems in the code of the plugin without having to re-bundle the Java classes of the plugin in a JAR library, follow these steps:

1. Download and unpack an *all platforms standalone version* of Oxygen XML Author/Editor/Developer.

 **Note:** The extracted folder name depends on which product variant you have downloaded. For the purpose of this procedure the folder will be referred to as [OXYGEN_DIR].

2. Set up the Oxygen SDK following *this set of instructions*.
3. Create an Eclipse Java Project (let's call it MyPluginProject) from one of the sample plugins (the Workspace Access plugin for example).
4. In the MyPluginProject folder, create a folder called myPlugin. In this new folder copy the plugin.xml from the sample plugin. Modify the added plugin.xml to add a library reference to the directory where Eclipse copies the compiled output. To find out where this directory is located, invoke the context menu of the project (in the **Project** view), and go to **Build Path > Configure Build Path...**. Then inspect the value of the **Default output folder** text box.

Example: If the compiled output folder is classes, then you need to add in the plugin.xml the following library reference:

```
<library name="..../classes"/>
```

5. Copy the plugin.dtd from the [OXYGEN_DIR]/plugins folder in the root MyPluginProject folder.
6. In the MyPluginProject's build path add external JAR references to all the JAR libraries in the [OXYGEN_DIR]/lib folder. Now your MyPluginProject should compile successfully.
7. In the Eclipse IDE, create a new *Java Application* configuration for debugging. Set the **Main class** box to ro.sync.exml.Oxygen. Click the **Arguments** tab and add the following code snippet in the **VM arguments** input box, making sure that the path to the plugins directory is the correct one:

```
-Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m
-XX:MaxPermSize=384m -Dcom.oxygenxml.editor.plugins.dir=D:\projects\MyPluginProject
```

 **Note:** If you need to configure the plugin for Oxygen XML Author or Oxygen XML Developer, set the com.oxygenxml.app.descriptor to ro.sync.exml.AuthorFrameDescriptor or ro.sync.exml.DeveloperFrameDescriptor, respectively.

8. Add a break point in the source of one of your Java classes.
9. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

Debugging an Oxygen SDK Extension Using the Eclipse Workbench

To debug problems in the extension code without having to bundle the extension's Java classes in a JAR library, perform the following steps:

1. Download and unpack an *all platforms standalone version* of Oxygen XML Author/Editor to a folder on your hard drive.

 **Note:** Name the folder [OXYGEN_DIR].

2. Create an Eclipse Java Project (let's call it MySDKProject) with the corresponding Java sources (for example a custom implementation of the `ro.sync.ecss.extensions.api.StylesFilter` interface).
3. In the Project's build path add external JAR references to all the JAR libraries in the `[OXYGEN_DIR]/lib` folder. Now your Project should compile successfully.
4. Start the standalone version of Oxygen from the `[OXYGEN_DIR]` and in the **Document Type Association** Preferences page edit the document type (for example **DITA**). In the **Classpath** tab, add a reference to your Project's `classes` directory and in the **Extensions** tab, select your custom `StylesFilter` extension as a value for the **CSS styles filter** property. Close the application to save the changes to the framework file.
5. Create a new Java Application configuration for debugging. The Main Class should be `ro.sync.exml.Oxygen`. The given VM Arguments should be

```
-Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m -XX:MaxPermSize=384m
```

6. Add a break point in one of the source Java classes.
7. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

Extending the Java Functionality of an Existing Framework (Document Type)

Question

How can I change the way DocBook 4 `xref`'s display in author view based on what element is at the `linkend`?

Please follow the steps below:

1. Create a Maven Java project and add a dependency on the oXygen classes:

```
<dependency>
  <groupId>com.oxygenxml</groupId>
  <artifactId>oxygen-sdk</artifactId>
  <version>${oxygen.version}</version>
</dependency>
```

where `${oxygen.version}` is the version of Oxygen XML Editor.

Alternatively, if the project does not use Maven, all the transitive dependencies of the above Maven artifact need to be added to the classpath of the project.

2. Also add to the project's class path the: "[OXYGEN_DIR]\frameworks\docbook\docbook.jar".
3. Create a class that extends `ro.sync.ecss.extensions.docbook.DocBook4ExtensionsBundle` and overwrites the method:
`ro.sync.ecss.extensions.api.ExtensionsBundle#createLinkTextResolver()`
4. For your custom resolver implementation you can start from the Java sources of the
`ro.sync.ecss.extensions.docbook.link.DocbookLinkTextResolver` (the Java code for the entire DocBook customization is present in a subfolder in the Author SDK).
5. Pack your extension classes in a JAR file. Copy the JAR to: "[OXYGEN_DIR]\frameworks\docbook\custom.jar".
6. Start Oxygen XML Editor.
7. *Open the Preferences dialog box* and go to **Document Type Association**. Edit the DocBook 4 document type. In the **Classpath** list add the path to the new JAR. In the extensions list select your custom extension instead of the regular DocBook one.
8. You can rename the document type and also the "docbook" framework folder to something else like "custom_docbook" and share it with others. A document type can also be installed using our [add-on support](#).

Controlling XML Serialization in the Author Component

Question

How can I force the Author Component to save the XML with zero indent size and not to break the line inside block-level elements?

Answer

Usually, in a standalone version of Oxygen XML Editor, the **Editor >Format** and **Editor > Format > XML** preferences pages allow you to control the way the XML is saved on the disk after you edit it in the **Author** mode.

In the editor application (Standalone or Eclipse-based), you can either bundle a *default set of options* or use the `PluginWorkspace.setGlobalObjectProperty(String, Object)` API:

```
//For not breaking the line
//Long line
pluginWorkspace.setObjectProperty("editor.line.width", new Integer(100000));
//Do not break before inline elements
pluginWorkspace.setObjectProperty("editor.format.indent.inline.elements", false);

//For forcing zero indent
//Force indent settings to be controlled by us
pluginWorkspace.setObjectProperty("editor.detect.indent.on.open", false);
//Zero indent size
pluginWorkspace.setObjectProperty("editor.indent.size.v9.2", 0);
```

In the Author Component, you can either bundle a *fixed set of options*, or use our Java API to set properties which overwrite the default options:

```
//For not breaking the line
//Long line
AuthorComponentFactory.getInstance().setObjectProperty("editor.line.width", new Integer(100000));
//Do not break before inline elements
AuthorComponentFactory.getInstance().setObjectProperty("editor.format.indent.inline.elements", false);

//For forcing zero indent
//Force indent settings to be controlled by us
AuthorComponentFactory.getInstance().setObjectProperty("editor.detect.indent.on.open", false);
//Zero indent size
AuthorComponentFactory.getInstance().setObjectProperty("editor.indent.size.v9.2", 0);
```

How can I add a custom Outline view for editing XML documents in the Text mode?

Let's say you have XML documents like

```
<doc startnumber="15">
  <sec counter="no">
    <info/>
    <title>Introduction</title>
  </sec>
  <sec>
    <title>Section title</title>
    <para>Content</para>
    <sec>
      <title>Section title</title>
      <para>Content</para>
    </sec>
  </sec>
  <sec>
    <title>Section title</title>
    <para>Content</para>
  </sec>
</doc>
```

and you want to display the XML content in a simplified Outline view like:

```
doc "15"
sec Introduction
sec 15 Section title
sec 15.1 Section title
sec 16 Section title
```

Usually an Outline should have the following characteristics:

1. Double clicking in the Outline the corresponding XML content would get selected.
2. When the caret moves in the opened XML document the Outline would select the proper entry.
3. When modifications occur in the document, the Outline would refresh.

A simple implementation using a Workspace Access plugin type could be something like:

```
/** 
 * Simple Outline for the Text mode based on executing XPaths over the text content.
```

```

/*
public class CustomWorkspaceAccessPluginExtension implements WorkspaceAccessPluginExtension {
    /**
     * The custom outline list.
     */
    private JList customOutlineList;

    /**
     * Maps outline nodes to ranges in document
     */
    private WSXMLTextNodeRange[] currentOutlineRanges;

    /**
     * The current text page
     */
    private WSXMLTextEditorPage currentTextPage;

    /**
     * Disable caret listener when we select from the caret listener.
     */
    private boolean enableCaretListener = true;

    /**
     * @see
ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension#applicationStarted(ro.sync.exml.workspace.api.standalone.StandalonePluginWorkspace)

     */
    @Override
    public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
        pluginWorkspaceAccess.addViewComponentCustomizer(new ViewComponentCustomizer() {
            /**
             * @see
ro.sync.exml.workspace.api.standalone.ViewComponentCustomizer#customizeView(ro.sync.exml.workspace.api.standalone.ViewInfo)

            */
            @Override
            public void customizeView(ViewInfo viewInfo) {
                if(
                    //The view ID defined in the "plugin.xml"
                    "SampleWorkspaceAccessID".equals(viewInfo.getViewID())) {
                    customOutlineList = new JList();
                    //Render the content in the Outline.
                    customOutlineList.setCellRenderer(new DefaultListCellRenderer() {
                        /**
                         * @see javax.swing.DefaultListCellRenderer#getListCellRendererComponent(javax.swing.JList,
java.lang.Object, int, boolean, boolean)
                         */
                        @Override
                        public Component getListCellRendererComponent(JList<?> list, Object value, int index,
                            boolean isSelected, boolean cellHasFocus) {
                            JLabel label = (JLabel) super.getListCellRendererComponent(list, value, index, isSelected,
                                cellHasFocus);
                            String val = null;
                            if(value instanceof Element) {
                                Element element = ((Element)value);
                                val = element.getNodeName();
                                if(!"".equals(element.getAttribute("startnumber"))) {
                                    val += " " + "" + element.getAttribute("startnumber") + "";
                                }
                                NodeList titles = element.getElementsByTagName("title");
                                if(titles.getLength() > 0) {
                                    val += " \" " + titles.item(0).getTextContent() + "\"";
                                }
                            }
                            label.setText(val);
                            return label;
                        }
                    });
                    //When we click a node, select it in the text page.
                    customOutlineList.addMouseListener(new MouseAdapter() {
                        @Override
                        public void mouseClicked(MouseEvent e) {
                            if(SwingUtilities.isLeftMouseButton(e) && e.getClickCount() == 2) {
                                int sel = customOutlineList.getSelectedIndex();
                                enableCaretListener = false;
                                try {
                                    currentTextPage.select(currentTextPage.getOffsetOfLineStart(currentOutlineRanges[sel].getStartLine()) +
                                        currentOutlineRanges[sel].getStartColumn() - 1,
                                        currentTextPage.getOffsetOfLineStart(currentOutlineRanges[sel].getEndLine()) +
                                        currentOutlineRanges[sel].getEndColumn());
                                } catch (BadLocationException e1) {
                                    e1.printStackTrace();
                                }
                                enableCaretListener = true;
                            }
                        }
                    });
                    viewInfo.setComponent(new JScrollPane(customOutlineList));
                }
            }
        });
    }
}

```

```

        viewInfo.setTitle("Custom Outline");
    }
}

pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
     */
    @Override
    public void editorOpened(URL editorLocation) {
        //An editor was opened
        WSEditor editorAccess = pluginWorkspaceAccess.getEditorAccess(editorLocation,
StandalonePluginWorkspace.MAIN_EDITING_AREA);
        if(editorAccess != null) {
            WSEditorPage currentPage = editorAccess.getCurrentPage();
            if(currentPage instanceof WSXMLTextEditorPage) {
                //User editing in Text mode an opened XML document.
                final WSXMLTextEditorPage xmlTP = (WSXMLTextEditorPage) currentPage;
                //Reconfigure outline on each change.
                xmlTP.getDocument().addDocumentListener(new DocumentListener() {
                    @Override
                    public void removeUpdate(DocumentEvent e) {
                        reconfigureOutline(xmlTP);
                    }
                    @Override
                    public void insertUpdate(DocumentEvent e) {
                        reconfigureOutline(xmlTP);
                    }
                    @Override
                    public void changedUpdate(DocumentEvent e) {
                        reconfigureOutline(xmlTP);
                    }
                });
                JTextArea textComponent = (JTextArea) xmlTP.getTextComponent();
                textComponent.addCaretListener(new CaretListener() {
                    @Override
                    public void caretUpdate(CaretEvent e) {
                        if(currentOutlineRanges != null && currentTextPage != null && enableCaretListener) {
                            enableCaretListener = false;
                            //Find the node to select in the outline.
                            try {
                                int line = xmlTP.getLineOfOffset(e.getDot());
                                for (int i = currentOutlineRanges.length - 1; i >= 0; i--) {
                                    if(line > currentOutlineRanges[i].getStartLine() && line <
currentOutlineRanges[i].getEndLine()) {
                                        customOutlineList.setSelectedIndex(i);
                                        break;
                                    }
                                }
                            } catch (BadLocationException e1) {
                                e1.printStackTrace();
                            }
                            enableCaretListener = true;
                        }
                    }
                });
            }
        }
    }
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorActivated(java.net.URL)
     */
    @Override
    public void editorActivated(URL editorLocation) {
        //An editor was selected, reconfigure the common outline
        WSEditor editorAccess = pluginWorkspaceAccess.getEditorAccess(editorLocation,
StandalonePluginWorkspace.MAIN_EDITING_AREA);
        if(editorAccess != null) {
            WSEditorPage currentPage = editorAccess.getCurrentPage();
            if(currentPage instanceof WSXMLTextEditorPage) {
                //User editing in Text mode an opened XML document.
                WSXMLTextEditorPage xmlTP = (WSXMLTextEditorPage) currentPage;
                reconfigureOutline(xmlTP);
            }
        }
    }
}, StandalonePluginWorkspace.MAIN_EDITING_AREA);

/***
 * Reconfigure the outline
 *
 * @param xmlTP The XML Text page.
 */
protected void reconfigureOutline(final WSXMLTextEditorPage xmlTP) {
    try {
        //These are DOM nodes.
        Object[] evaluateXPath = xmlTP.evaluateXPath("//doc | //sec");
    }
}

```

```

//These are the ranges each node takes in the document.
currentOutlineRanges = xmlTP.findElementsByXPath("//doc //sec");
currentTextPage = xmlTP;
DefaultListModel listModel = new DefaultListModel();
if(evaluateXPath != null) {
    for (int i = 0; i < evaluateXPath.length; i++) {
        listModel.addElement(evaluateXPath[i]);
    }
}
customOutlineList.setModel(listModel);
} catch(XPathException ex) {
    ex.printStackTrace();
}
}

/**
 * @see ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension#applicationClosing()
 */
@Override
public boolean applicationClosing() {
    return true;
}
}

```

Dynamically Adding Form Controls Using a StylesFilter

Usually, a form control is added from the CSS using the [oxy_editor\(\)](#) function. However, in some cases you don't have all the information you need to properly initialize the form control at CSS level. In these cases you can add the form controls by using the API, more specifically [ro.sync.ecss.extensions.api.StylesFilter](#).

For instance, let's assume that we want a combo box form control and the values to populate the combo are specified inside a file (for a more interesting scenario we could imagine that they come from a database). Here is how to add the form control from the API:

```

public class SDFStylesFilter implements StylesFilter {

    public Styles filter(Styles styles, AuthorNode authorNode) {
        if(authorNode.getType() == AuthorNode.NODE_TYPE_PSEUDO_ELEMENT
            && "before".equals(authorNode.getName())) {
            authorNode = authorNode.getParent();
        if ("country".equals(authorNode.getName())) {
            // This is the BEFORE pseudo element of the "country" element.
            // Read the supported countries from the configuration file.
            Map<String, Object> formControlArgs = new HashMap<String, Object>();
            formControlArgs.put(ImplacementEditorArgumentKeys.PROPERTY_EDIT, "#text");
            formControlArgs.put(ImplacementEditorArgumentKeys.PROPERTY_TYPE, ImplacementEditorArgumentKeys.TYPE_COMBOBOX);

            // This will be a comma separated enumeration: France, Spain, Great Britain
            String countries = readCountriesFromFile();
            formControlArgs.put(ImplacementEditorArgumentKeys.PROPERTY_VALUES, countries);
            formControlArgs.put(ImplacementEditorArgumentKeys.PROPERTY_EDITABLE, "false");

            // We also add a label in form of the form control.
            Map<String, Object> labelProps = new HashMap<String, Object>();
            labelProps.put("text", "Country: ");
            labelProps.put("style", "* {width: 100px; color: gray;}");
            StaticContent[] mixedContent = new StaticContent[] {new LabelContent(labelProps), new
            EditorContent(formControlArgs)};
            styles.setProperty(Styles.KEY_MIXED_CONTENT, mixedContent);
        }
        }

        // The previously added form control is the only way the element can be edited.
        if ("country".equals(authorNode.getName())) {
            styles.setProperty(Styles.KEY_VISIBILITY, "-oxy-collapse-text");
        }

        return styles;
    }
}

```

If the execution of the `formControlArgs.put(ImplacementEditorArgumentKeys.PROPERTY_VALUES, countries);` line consumes too much execution time (for example if it connects to a database or if it needs to extract data from a very large file), you can choose to delay it until the values are actually needed by the form control. This approach is called *lazy evaluation* and can be implemented as follows:

```

formControlArgs.put(ImplacementEditorArgumentKeys.PROPERTY_VALUES, new LazyValue<List<CIValue>>() {
    public java.util.List<CIValue> get() {
        // We avoid reading the possible values until they are actually requested.
        // This will be a List with CIValues created over countries: France, Spain, Great Britain
    }
}

```

```

        return readCountriesFromFile();
    }
});

```

The lazy evaluation approach can be used for the following form controls properties:

- InplaceEditorArgumentKeys.PROPERTY_VALUES
- InplaceEditorArgumentKeys.PROPERTY_LABELS
- InplaceEditorArgumentKeys.PROPERTY_TOOLTIPS

The full source code for this example is available inside the [Author SDK](#).

Modifying the XML content on Open

Question

I have a bunch of DITA documents which have a fixed path the image `src` attributes. These paths are not valid and I am trying to move away from this practice by converting it in to relative paths. When an XML document is opened, can I trigger the Java API to change the fixed path to a relative path?

Answer

Our Plugins SDK:http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins contains a sample Plugin Type called `WorkspaceAccess`. Such a plugin is notified when the application starts and it can do what you want in a couple of ways:

1. You add a listener which notifies you when the user opens an XML document. Then if the XML document is opened in the Author visual editing mode you can use our Author API to change attributes:

```

pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
     */
    @Override
    public void editorOpened(URL editorLocation) {
        WSEditor openedEditor =
            pluginWorkspaceAccess.getCurrentEditorAccess(StandalonePluginWorkspace.MAIN_EDITING_AREA);
        if(openedEditor.getCurrentPage() instanceof WSAuthorEditorPage) {
            WSAuthorEditorPage authPage = (WSAuthorEditorPage) openedEditor.getCurrentPage();
            AuthorDocumentController docController = authPage.getDocumentController();
            try {
                //All changes will be undone by pressing Undo once.
                docController.beginCompoundEdit();
                fixupImageRefs(docController,
                    docController.getAuthorDocumentNode());
            } finally {
                docController.endCompoundEdit();
            }
        }
    }

    private void fixupImageRefs(AuthorDocumentController docController, AuthorNode authorNode) {
        if(authorNode instanceof AuthorParentNode) {
            //Recurse
            List<AuthorNode> contentNodes = ((AuthorParentNode)authorNode).getContentNodes();
            if(contentNodes != null) {
                for (int i = 0; i < contentNodes.size(); i++) {
                    fixupImageRefs(docController, contentNodes.get(i));
                }
            }
        }
        if(authorNode.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
            AuthorElement elem = (AuthorElement) authorNode;
            if("image".equals(elem.getLocalName())) {
                if(elem.getAttribute("href") != null) {
                    String originalHref = elem.getAttribute("href").getValue();
                    URL currentLocation = docController.getAuthorDocumentNode().getXMLBaseURL();
                    //TODO here you compute the new href.
                    String newHref = null;
                    docController.setAttribute("href", new AttrValue(newHref), elem);
                }
            }
        }
    }
},
StandalonePluginWorkspace.MAIN_EDITING_AREA);

```

2. You also have API to open XML documents in the application:

```
ro.sync.exml.workspace.api.Workspace.open(URL)
```

So you can create up a plugin which automatically opens one by one XML documents from a certain folder in the application, makes modifications to them, saves the content by calling:

```
ro.sync.exml.workspace.api.editor.WSEditorBase.save()
```

and then closes the editor:

```
ro.sync.exml.workspace.api.Workspace.close(URL)
```

Modifying the XML content on Save

Question

Is it possible to get Oxygen to update the revised date on a DITA document when it's saved?

Answer

Our Plugins SDK:http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins contains a sample Plugin Type called *WorkspaceAccess*. Such a plugin is notified when the application starts.

You can add a listener which notifies you before the user saves an XML document. Then if the XML document is opened in the Author visual editing mode you can use our Author API to change attributes before the save takes place:

```
@Override
public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
    pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
        //An editor was opened
        @Override
        public void editorOpened(URL editorLocation) {
            final WSEditor editorAccess = pluginWorkspaceAccess.getEditorAccess(editorLocation,
                PluginWorkspace.MAIN_EDITING_AREA);
            if(editorAccess != null){
                editorAccess.addEditorListener(new ro.sync.exml.workspace.api.listeners.WSEditorListener(){
                    //Editor is about to be saved
                    @Override
                    public boolean editorAboutToBeSavedVeto(int operationType) {
                        if(EditorPageConstants.PAGE_AUTHOR.equals(editorAccess.getCurrentPageID())){
                            WSAuthorEditorPage authorPage = (WSAuthorEditorPage) editorAccess.getCurrentPage();
                            AuthorDocumentController controller = authorPage.getDocumentController();
                            try {
                                //Find the revised element
                                AuthorNode[] nodes = controller.findNodesByXPath("//revised", true, true, true);
                                if(nodes != null && nodes.length > 0){
                                    AuthorElement revised = (AuthorElement) nodes[0];
                                    //Set the modified attribute to it...
                                    controller.setAttribute("modified", new AttrValue(new Date().toString()), revised);
                                }
                            } catch (AuthorOperationException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                    //And let the save continue..
                    return true;
                });
            }
        }
    }, PluginWorkspace.MAIN_EDITING_AREA);
}
```

Save a new document with a predefined file name pattern

Question

Is it possible to get Oxygen Author to automatically generate a file name comprising a UUID plus file extension using the SDK?

Answer

This could be done implementing a plugin for Oxygen XML Editor using our Plugins SDK:

http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

There is a type of plugin called *Workspace Access* that can be used to add a listener to be notified before an opened editor is saved. The implemented plugin would intercept the save events when a newly created document is untitled and display an alternative chooser dialog box, then save the topic with the proper name.

The Java code for this would look like:

```

private static class CustomEdListener extends WSEditorListener{
    private final WSEditor editor;
    private final StandalonePluginWorkspace
        pluginWorkspaceAccess;
    private boolean saving = false;
    public CustomEdListener(StandalonePluginWorkspace pluginWorkspaceAccess, WSEditor editor) {
        this.pluginWorkspaceAccess = pluginWorkspaceAccess;
        this.editor = editor;
    }
    @Override
    public boolean editorAboutToBeSavedVeto(int operationType) {
        if(! saving &&
            editor.getEditorLocation().toString().contains("Untitled")) {
            File chosenDir = pluginWorkspaceAccess.chooseDirectory();
            if(chosenDir != null) {
                final File chosenFile = new File(chosenDir, UUID.randomUUID().toString() + ".dita");
                SwingUtilities.invokeLater(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            saving = true;
                            editor.saveAs(new URL(chosenFile.toURI().toASCIIString()));
                        } catch (MalformedURLException e) {
                            e.printStackTrace();
                        } finally {
                            saving = false;
                        }
                    }
                });
            }
        }
        //Reject the original save request.
        return false;
    }
    return true;
}
@Override
public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
    pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
        @Override
        public void editorOpened(URL editorLocation) {
            final WSEditor editor = pluginWorkspaceAccess.getEditorAccess(editorLocation,
                PluginWorkspace.MAIN_EDITING_AREA);
            if(editor != null && editor.getEditorLocation().toString().contains("Untitled")) {
                //Untitled editor
                editor.addEditorListener(new CustomEdListener(pluginWorkspaceAccess, editor));
            }
        }
    },
    PluginWorkspace.MAIN_EDITING_AREA);
.....

```

Auto-generate an ID when a document is opened or created

Question

Is it possible to configure how the application generates ids? For project compliance we need ids having a certain format for each created topic.

Answer

This could be done implementing a plugin for Oxygen XML Editor using our Plugins SDK:

http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

There is a type of plugin called "Workspace Access" which can be used to add a listener to be notified when an editor is opened.

The implemented plugin would intercept the editor opened and editor page changed events (which occur when a new editor is created) and generate a new ID attribute value on the root element.

The Java code for this would look like:

```

pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
     */
    @Override
    public void editorOpened(URL editorLocation) {
        WSEditor ed = pluginWorkspaceAccess.getEditorAccess(editorLocation, PluginWorkspace.MAIN_EDITING_AREA);

        generateID(ed);
    }
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorPageChanged(java.net.URL)
     */
    @Override
    public void editorPageChanged(URL editorLocation) {
        WSEditor ed = pluginWorkspaceAccess.getEditorAccess(editorLocation, PluginWorkspace.MAIN_EDITING_AREA);

        generateID(ed);
    }

    private void generateID(WSEditor ed) {
        if(ed.getCurrentPage() instanceof WSAuthorEditorPage) {
            WSAuthorEditorPage authorEditPage = (WSAuthorEditorPage) ed.getCurrentPage();
            AuthorDocumentController ctrl = authorEditPage.getDocumentController();
            AuthorElement root = ctrl.getAuthorDocumentNode().getRootElement();
            if(root.getAttribute("id") == null || !root.getAttribute("id").getValue().startsWith("generated_"))
            {
                ctrl.setAttribute("id", new AttrValue("generated_" + Math.random()), root);
            }
        }
    }
}, PluginWorkspace.MAIN_EDITING_AREA);

```

Use a custom view with the Oxygen XML Editor distribution

Question

Is it possible to create a custom view in Eclipse which can insert certain XML fragments in the documents opened with the Oxygen XML Editor?

Answer

Here you can find more information about the Eclipse part of the oXygen SDK:

http://www.oxygenxml.com/oxygen_sdk.html#oXygen_Eclipse_plugin

Use the provided Oxygen XML Editor sample project as a starting point. From any custom view/component you can have singleton access to the using the

`ro.sync.exml.workspace.api.PluginWorkspaceProvider.getPluginWorkspace()` API.

The Java code for inserting a certain XML fragment in the currently open editor (either in the Text or Author editing modes) would look like this:

```

WSEditor currentEditorAccess =
PluginWorkspaceProvider.getPluginWorkspace().getCurrentEditorAccess(PluginWorkspace.MAIN_EDITING_AREA);
if(currentEditorAccess.getCurrentPage() instanceof WSXMLTextEditorPage) {
    //Editor opened in Text page
    WSXMLTextEditorPage tp = (WSXMLTextEditorPage) currentEditorAccess.getCurrentPage();
    //You can access an API to insert text in the XML content
    tp.getDocument().insertString(tp.getCaretOffset(), "<testTag/>", null);
    //This is the internal StyledText implementation
    tp.getTextComponent()
    //You can use this XPath API to find the range of an XML element.
    tp.findElementsByXPath(xpathExpression)
} else if(currentEditorAccess.getCurrentPage() instanceof WSAuthorEditorPage) {
    //Editor opened in Author page
    try {

```

```
WSAuthorEditorPage authPage = (WSAuthorEditorPage) currentEditorAccess.getCurrentPage();
//Then you can do stuff like this to insert XML at caret position
//    authPage.getDocumentController().insertXMLFragment("<testTag/>", authPage.getCaretOffset());
//} catch (AuthorOperationException e) {
//    // TODO Auto-generated catch block
//    e.printStackTrace();
//}
}
```

Chapter

11

Transforming Documents

Topics:

- *Transformation Scenarios*
- *Output Formats*

XML documents can be transformed into a variety of user-friendly output formats that can be viewed by other users. This process is known as a *transformation*.

Transformation Scenarios

A transformation scenario is a set of complex operations and settings that gives you the possibility to obtain outputs of multiple types (XML, HTML, PDF, EPUB, etc.) from the same source of XML files and stylesheets.

Executing a transformation scenario implies multiple actions, such as:

- Validating the input file.
- Obtaining intermediate output files (for example, formatting objects for the XML to PDF transformation).
- Using transformation engines to produce the output.

Before transforming an XML document in Oxygen XML Editor, you need to define a transformation scenario to apply to that document. A scenario is a set of values for various parameters that define a transformation. It is not related to a particular document, but rather to a document type. Types of transformation scenarios include:

- **Scenarios that Apply to XML Files** - This type of scenario contains the location of an XSLT stylesheet that is applied on the edited XML document, as well as other transformation parameters.
- **Scenarios that Apply to XSLT Files** - This type of scenario contains the location of an XML document, on which the edited XSLT stylesheet is applied, as well as other transform parameters.
- **Scenarios that Apply to XQuery Files** - This type of scenario contains the location of an XML source, on which the edited XQuery file is applied, as well as other transform parameters. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()`. When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- **Scenarios that Apply to SQL Files** - This type of scenario specifies a database connection for the database server that runs the SQL file that is associated with the scenario. The data processed by the SQL script is located in the database.
- **Scenarios that Apply to XProc Files** - This type of scenario contains the location of an XProc script, as well as other transform parameters.
- **DITA-OT Scenarios** - This type of scenario provides the parameters for an ANT transformation that executes a DITA-OT build script. Oxygen XML Editor comes with a built-in version of ANT and a built-in version of DITA-OT, although you can also set other versions in the scenario.
- **ANT Scenarios** - This type of scenario contains the location of an ANT build script, as well as other transform parameters.



Note:

Status messages generated during the transformation process are displayed in the [Information view](#).

Defining a New Transformation Scenario

Defining a transformation scenario is the first step in the process of transforming a document. The following types of scenarios are available:

- ***XML Transformation with XSLT*** - Specifies the transformation parameters and location of an XSLT stylesheet that is applied to the edited XML document. This scenario is useful when you develop an XML document and the XSLT document is in its final form.
- ***XML Transformation with XQuery*** - Specifies the transform parameters and location of an XQuery file that is applied to the edited XML document.
- ***DITA-OT Transformation*** - Specifies the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Editor comes with a built-in version of Ant and a built-in version of DITA-OT but different versions can be set in the scenario.
- ***ANT Transformation*** - Allows you to configure the options and parameters of an ANT build script.
- ***XSLT Transformation*** - Specifies the transformation parameters and location of an XML document to which the edited XSLT stylesheet is applied. This scenario is useful when you develop an XSLT document and the XML document is in its final form.
- ***XProc Transformation*** - Specified the transformation parameters and location of an XProc script.

- **XQuery Transformation** - Specifies the transformation parameters and location of an XML source to which the edited XQuery file is applied. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()`. When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- **SQL Transformation** - Specifies a database connection for the database server that runs the SQL file associated with the scenario. The data processed by the SQL script is located in the database.

XML transformation with XSLT

To create an **XML transformation with XSLT** scenario, use one of the following methods:

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XSLT**.
- Use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XML transformation with XSLT**.
- Use the  **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XML transformation with XSLT**.



Note: If a scenario is already associated with the edited document, selecting  **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs:

- **XSLT**.
- **FO Processors**.
- **Output**.

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

Note: If the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and *the name of an initial template* is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty if you use *external XSLT processors*. Otherwise, a value is mandatory in the **XML URL** field.
- **XSL URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

 **Insert Editor Variables**

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor editor variables* or *custom editor variables* in the XML URL field.

 **Browse for local file**

Opens a local file browser dialog box allowing you to select a local file.

 **Browse for remote file**

Opens an URL browser dialog box allowing you to select a remote file.

 **Browse for archived file**

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

 **Browse Data Source Explorer**

Opens the *Data Source Explorer* window.

 **Search for file**

Allows you to find a file in the current project.

 **Open in editor**

Opens the file in an editor panel with the path that is specified in the **XML URL** text box.

The rest of the options available in the **XSLT** tab allow you to further customize the transformation scenario:

- **Use "xmlstylesheet" declaration** - Use the stylesheet declared with an `xmlstylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default, this checkbox is not selected and the transformation applies the XSLT stylesheet that is specified in the **XSL URL** field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xmlstylesheet` processing instruction.
- **Transformer** - This drop-down list presents all the transformation engines available to Oxygen XML Editor for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).
-  **Advanced options** - Allows you to configure the advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the *Saxon-HE/PE/EE preferences page*. For the current transformation scenario, these **Advanced options** override the options configured in the *Saxon-HE/PE/EE preferences page*. The **Initial mode and template** option is only available in the **Advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template that starts the XSLT transformation or the initial mode of the transformation.
- **Parameters** - Opens *the Configure parameters dialog*, allowing you to configure the XSLT parameters used in the current transformation. In this dialog you can also configure the parameters of additional stylesheets by using the **Additional XSLT stylesheets** button. If the XSLT transformation engine is custom-defined, you can not use this dialog to configure the parameters sent to the custom engine. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line that starts the transformation process.
- **Extensions** - Opens *the dialog for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.
- **Additional XSLT stylesheets** - Opens *the dialog for adding XSLT stylesheets* that are applied on the main stylesheet that is specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

Configure XSLT Parameters

The global parameters of the XSLT stylesheet used in a transformation scenario can be configured by using the **Parameters** button in the **XSLT** tab of a new or edited transformation scenario dialog.

The table displays all the parameters of the current XSLT stylesheet, all imported and included stylesheets, and all *additional stylesheets*, along with their descriptions and current values. You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

If the **XPath** column is checked, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.

For example, you can use expressions such as:

```
doc('test.xml')//entry
//person[@attr='val']
```



Note:

1. The **doc** function solves the argument relative to the XSL stylesheet location. You can use full paths or editor variables (such as \${cfdu} [current file directory]) to specify other locations:
`doc(' ${cfdu} /test.xml')/*`
2. You cannot use XSLT Functions. Only XPath functions are allowed.

The following actions are available for managing the parameters:

New

Opens the **Add Parameter** dialog that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is enabled, the parameter will be evaluated as an XPath expression.

Edit

Opens the **Edit Parameter** dialog that allows you to edit the selected parameter. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is enabled, the parameter will be evaluated as an XPath expression.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The bottom panel presents the following:

- The default value of the parameter selected in the table.
- A description of the parameter, if available.
- The system ID of the stylesheet that declares it.

XSLT/XQuery Extensions

The **Libraries** dialog box is used to specify the jars and classes that contain extension functions called from the XSLT or XQuery file of the current transformation scenario.

An extension function called from the XSLT or XQuery file of the current transformation scenario will be searched, in the specified extensions, in the order of the list displayed in this dialog. To change the order of the items, select the item to be moved and press the **Move up** or **Move down** buttons.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XSLT result as input** - The FO processor is applied to the result of the XSLT transformation that is defined in the **XSLT** tab.

- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the  **Insert Editor Variables** button..
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the *Acrobat Reader* application).



Note: To set the web browser that is used for displaying HTML/XHTML pages, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor opens the file specified here. The file path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the  **Insert Editor Variables** button.
- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
 - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.
 - Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.
 - **XML** - If this is checked, Oxygen XML Editor displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
 - **SVG** - If this is checked, Oxygen XML Editor displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

Additional XSLT Stylesheets

The list of additional XSLT stylesheets can be edited in the dialog box opened by the **Additional XSLT Stylesheets** button in the **XSLT** tab of a new or edited transformation scenario dialog box. The following actions are available:

Add

Adds a stylesheet in the **Additional XSLT stylesheets** list using a file browser dialog box. You can type an *editor variable* in the file name field of the browser dialog box. The name of the stylesheet will be added in the list after the current selection.

Remove

Deletes the selected stylesheet from the **Additional XSLT stylesheets** list.

Open

Opens the selected stylesheet in a separate view.

Up

Moves the selected stylesheet up in the list.

Down

Moves the selected stylesheet down in the list.

XML Transformation with XQuery

Use the **XML transformation with XQuery** scenario to apply a transformation in which an XQuery file queries an XML file for the output results.

To create an **XML transformation with XQuery** scenario, use one of the following methods:

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XQuery**.
- Use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XML transformation with XQuery**.
- Use the  **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XML transformation with XQuery**.



Note: If a scenario is already associated with the edited document, selecting  **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs:

- **XQuery**.
- **FO Processor**.
- **Output**.

The XQuery Tab

The **XQuery** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.



Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

- **XQuery URL** - specifies the source XQuery file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

 **Insert Editor Variables**

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor editor variables* or *custom editor variables* in the XML URL field.

 **Browse for local file**

Opens a local file browser dialog box allowing you to select a local file.

 **Browse for remote file**

Opens an URL browser dialog box allowing you to select a remote file.

 **Browse for archived file**

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

 **Browse Data Source Explorer**

Opens the *Data Source Explorer* window.

 **Search for file**

Allows you to find a file in the current project.

 **Open in editor**

Opens the file in an editor panel with the path that is specified in the **XML URL** text box.

The rest of the options available in the **XQuery** tab allow you to further customize the transformation scenario:

- **Transformer** - This drop-down list presents all the transformation engines available to Oxygen XML Editor for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).;
-  **Advanced options** - *configure advanced options specific for the Saxon HE / PE / EE engine*.
- **Parameters** - Opens the **Configure parameters** dialog for configuring the XQuery parameters. You can use buttons in this dialog you can add, edit, or remove parameters. If the XQuery transformation engine is custom-defined you can not use this dialog to set parameters. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XQuery engine to include the necessary parameters in the command line that starts the transformation process.



Note: Use the **Filter** text box to search for a specific term in the entire parameters collection.

- **Extensions** - Opens *the dialog for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.

XSLT/XQuery Extensions

The **Libraries** dialog box is used to specify the jars and classes that contain extension functions called from the XSLT or XQuery file of the current transformation scenario.

An extension function called from the XSLT or XQuery file of the current transformation scenario will be searched, in the specified extensions, in the order of the list displayed in this dialog. To change the order of the items, select the item to be moved and press the  **Move up** or  **Move down** buttons.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.

- **XQuery result as input** - the FO processor is applied to the result of the XQuery transformation defined in the **XQuery** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Present as a sequence** - Enabling this option will reduce the time necessary to fetch the full result, as it will only fetch the first chunk of the result.
- **Prompt for file** - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the  **Insert Editor Variables** button..
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the *Acrobat Reader* application).



Note: To set the web browser that is used for displaying HTML/XHTML pages, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor opens the file specified here. The file path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the  **Insert Editor Variables** button.
- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
 - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.
 -  **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.
 - **XML** - If this is checked, Oxygen XML Editor displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
 - **SVG** - If this is checked, Oxygen XML Editor displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image.
 - **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

DITA OT Transformation

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**.
 - Use the **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **DITA OT Transformation**.
 - Use the **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **DITA OT Transformation**.
- Note:** If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **DITA Transformation Type** dialog box that presents the list of possible outputs.

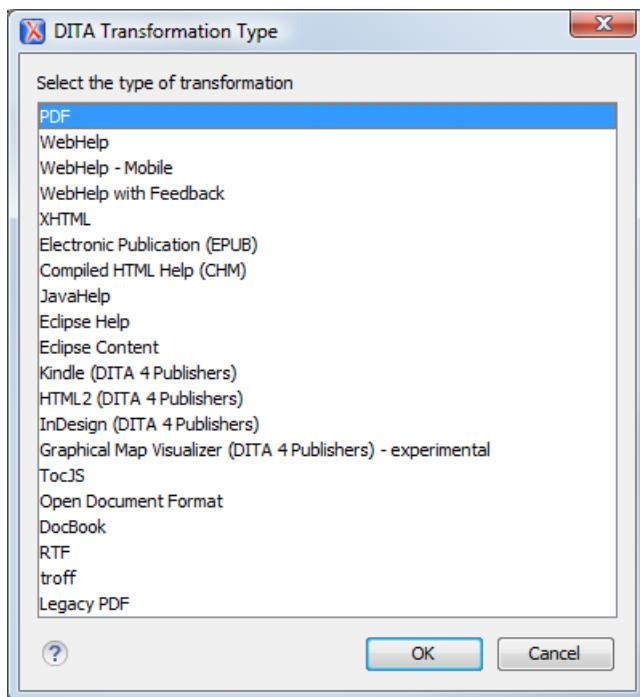


Figure 276: DITA Transformation Type Dialog Box

Select the desired type of output and click **OK**. This opens the **New Scenario** dialog box, which allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs (only those that are appropriate for the chosen output type will be displayed):

- **Skins** (Available for **WebHelp** and **WebHelp with Feedback** output types).
- **FO Processor** (Available for **PDF** output types).

- [Parameters](#)
- [Filters](#)
- [Advanced](#)
- [Output](#)

For information on creating an entirely new DITA OT transformation, see [Creating a DITA OT Customization Plugin](#) on page 456 and [Installing a Plugin in the DITA Open Toolkit](#) on page 457.

The Skins Tab

A *skin* is a collection of CSS properties that can alter the look of the output by changing colors, font types, borders, margins, and paddings. This allows you to rapidly adapt the look and feel of the output for your organization.

Oxygen XML Editor provides a set of predefined *skins* for the **DITA Map WebHelp** and **DITA Map WebHelp with Feedback** transformation scenarios.

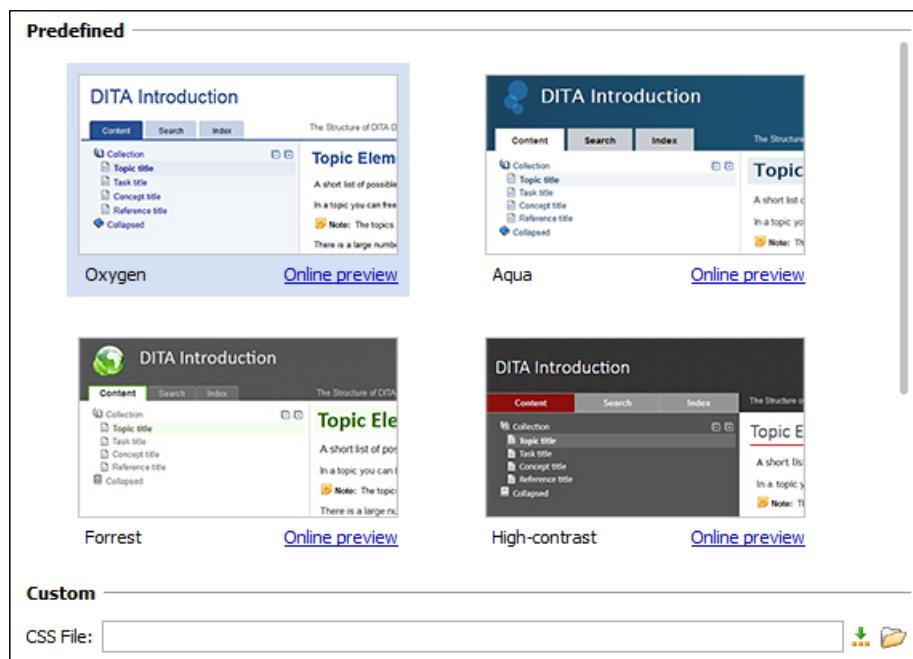


Figure 277: The Skins Tab

The predefined skins cover a wide range of chromatic themes, ranging from a very light one to a high-contrast variant. By default, the **Oxygen** skin is selected (notice the light blue border around the skin preview). If you want to obtain an output without any customization, deselect the currently selected skin.

To see how the *skin* looks when applied on a sample documentation project that is stored on the Oxygen XML Editor website, press the **Online preview** link.



Note: Press the **Create custom skin** link to open the [WebHelp Skin Builder](#) tool.

To further customize the look of the output, set the **CSS File** field to point to your custom CSS stylesheet or to a customized skin.



Note: A custom CSS file will overwrite a skin selection.



Note: The output can also be styled by setting the `args.css` parameter in the **Parameters tab**. The properties taken from the stylesheet referenced in this parameter take precedence over the properties declared in the skin set in the **Skins tab**.

The FO Processor Tab

This tab allows you to select an FO Processor, when you choose to generate PDF output.

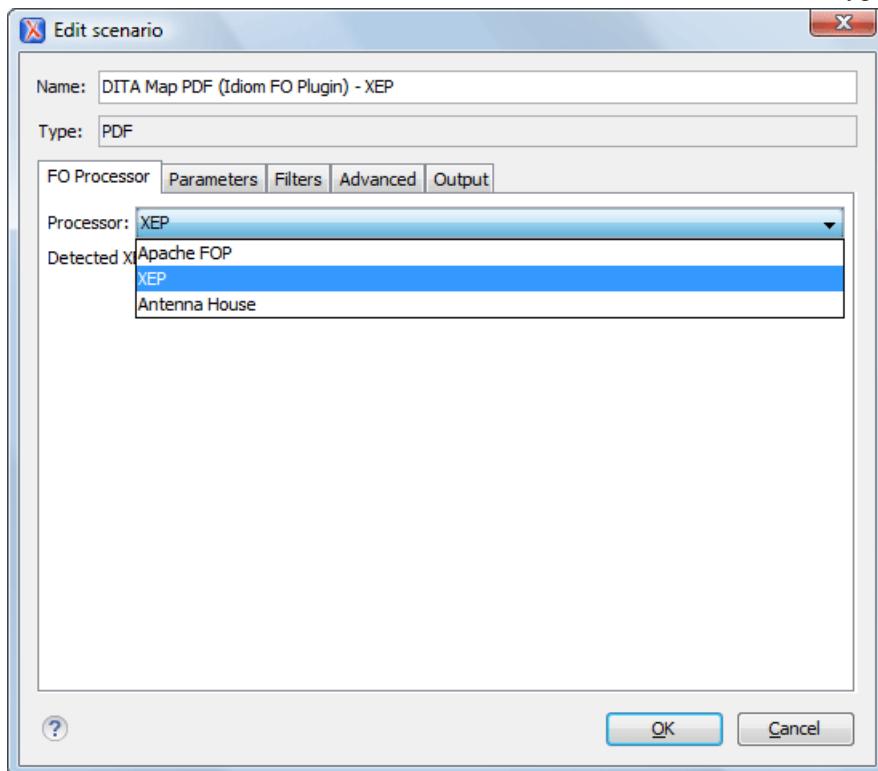


Figure 278: FO Processor Configuration Tab

You can choose the following processors:

- **Apache FOP** - The default processor that comes bundled with .
- **XEP** - The *RenderX* XEP processor.

If XEP is already installed, displays the detected installation path under the drop-down list.

XEP is considered installed if it was detected in one of the following sources:

- XEP was configured as an external FO Processor in the [FO Processors option page](#).
- The system property `com.oxygenxml.xep.location` was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows).
- XEP was installed in the
[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/lib directory of the installation directory.
- **Antenna House** - The [Antenna House](#) AH (v5) or XSL (v4) Formatter processor.

If Antenna House is already installed, displays the detected installation path under the drop-down list.

Antenna House is considered installed if it was detected in one of the following sources:

- Environment variable set by Antenna House installation (the newest installation version will be used, v5 being preferred over v4).
- Antenna House was added as an external FO Processor in the preferences pages.

To further customize the PDF output obtained from the Antenna House processor:

- **Edit** the transformation scenario.
- Open the [Parameters tab](#).
- Add the `env.AXF_OPT` parameter and point to Antenna House configuration file.

The Parameters Tab

The **Parameterstab** allows you to configure the parameters sent to the DITA-OT build file.

The table displays all the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: XHTML or PDF), along with their description and current values. You can find more information about each parameter in the [DITA OT Documentation](#). You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.



Note: To input parameter values at runtime, use the *ask editor variable* in the **Value** column.

The following actions are available for managing parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Filters Tab

The **Filters** tab allows you to add filters to remove certain content elements from the generated output.

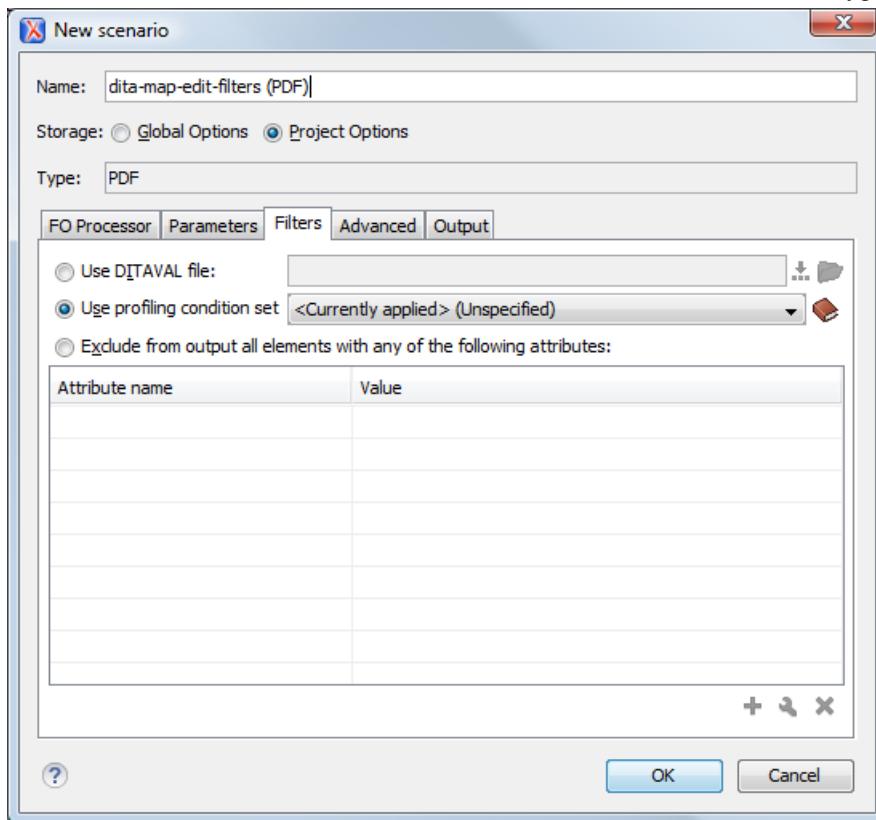


Figure 279: Edit Filters Tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the DITA map, you can specify the file to be used when filtering content. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button. You can find out more about constructing a DITAVAL file in the [DITA OT Documentation](#).
- **Use profiling condition set** - Sets the *profiling condition set* that will apply to your transformation.
- **Exclude from output all elements with any of the following attributes** - By using the **New**, **Edit**, or **Delete** buttons at the bottom of the pane, you can configure a list of attributes (name and value) to exclude all elements that contain any of these attributes from the output.

The Advanced Tab

The **Advanced** tab allows you to specify advanced options for the transformation scenario.

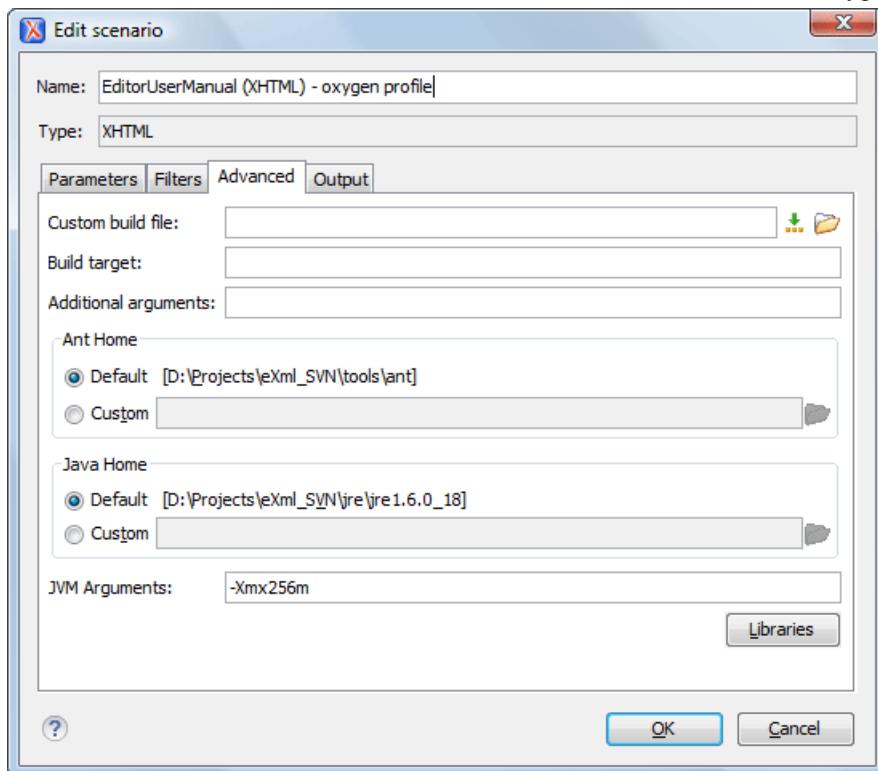


Figure 280: Advanced Settings Tab

You can specify the following parameters:

- **Custom build file** - If you use a custom DITA-OT build file, you can specify the path to the customized build file. If empty, the `build.xml` file from the `dita.dir` parameter that is configured in the **Parameters** tab is used. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button.
- **Build target** - Optionally, you can specify a build target for the build file. If no target is specified, the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation (such as `-verbose`).
- **Ant Home** - You can choose between the default or custom ANT installation to run the transformation. The default path can be configured in the [Ant preferences page](#).
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by .
- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by ANT. For example, if it is set to `-Xmx384m`, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (**OutOfMemoryError**).
- **Libraries** - By default, adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the ANT transformer.

The Output Tab

The **Output** tab allows you to configure options that are related to the location where the output is generated.

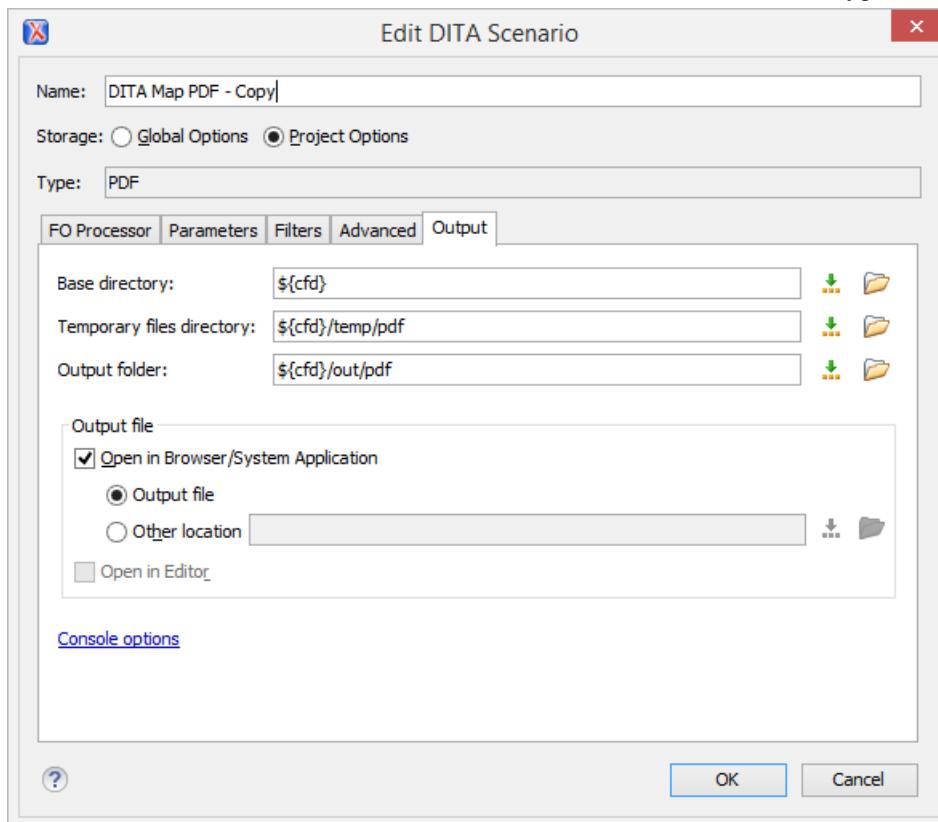


Figure 281: Output Settings Tab

You can specify the following parameters:

- **Base directory** - All the relative paths that appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Output folder** - The folder where the content of the final output is stored. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.

Note: If the DITA map or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

- **Open in Browser/System Application** - If enabled, Oxygen XML Editor automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the *Acrobat Reader* application).

Note: To set the web browser that is used for displaying HTML/XHTML pages, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor opens the file specified here. The file path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

At the bottom of the pane there is a link to the [Console options](#) preferences page that contains options to control the display of the console output received from the publishing engine.

Troubleshooting DITA Transformation Errors

If a DITA transformation results in errors or warnings, the information is displayed in the message panel at the bottom of the editor. The information includes the severity, description of the problem, the name of the resource, and the path of the resource.

To help prevent and solve DITA transformation problems, follow these steps:

1. [Validate the DITA map](#) by using the **Validate and Check for Completeness** action that is available on the **DITA Maps Manager** toolbar and in the **DITA Maps** menu.
2. If this action results in validation errors, solve them prior to executing the transformation. Also, you should pay attention to the warning messages because they may identify problems in the transformation.
3. [Execute the DITA transformation scenario](#).
4. If the transformation results in errors or warnings, they are displayed in the **Transformation problems** message panel at the bottom of the editor. The following information is presented to help you troubleshoot the problems:
 - **Severity** - The first column displays the following icons that indicate the severity of the problem:
 - **Informational** - The transformation encountered a condition of which you should be aware.
 - **Warning** - The transformation encountered a problem that should be corrected.
 - **Error** - The transformation encountered a more severe problem, and the output is affected or cannot be generated.
 - **Info** - You can click on the **See More** icon to open a web page that contains details about DITA-OT error messages.
 - **Description** - A description of the problem.
 - **Resource** - The name of the transformation resource.
 - **System ID** - The path of the transformation resource.
5. Use this information or other resources from the online DITA-OT community to solve the transformation problems before re-executing the transformation scenario.

ANT Transformation

An **ANT** transformation scenario is usually associated with an Ant build script. Oxygen XML Editor runs an **ANT** transformation scenario as an external process that executes the Ant build script with the built-in Ant distribution (Apache Ant version 1.8.2) that comes with the application, or optionally with a custom Ant distribution configured in the scenario.

To create an **ANT transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **ANT transformation**.
- Use the **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **ANT transformation**.
- Use the **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **ANT transformation**.



Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs:

- *The Options tab.*
- *The Parameters tab.*
- *The Output tab.*

The Options Tab

The **Options** tab allows you to specify the following options:

- **Working directory** - The path of the current directory of the Ant external process. An *editor variable* can be inserted for the file path by using the  **Insert Editor Variables** button.
- **Build file** - The Ant script file that is the input of the Ant external process. An *editor variable* can be inserted for the file path by using the  **Insert Editor Variables** button.
- **Build target** - Optionally, you can specify a build target for the Ant script file. If no target is specified, the Ant target that is specified as the default in the Ant script file is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation (such as -verbose).
- **Ant Home** - You can choose between the default or custom ANT installation to run the transformation. The default path can be configured in the [Ant preferences page](#).
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Editor.
- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by ANT. For example, if it is set to -Xmx384m, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (**OutOfMemoryError**).
- **Libraries** - By default, Oxygen XML Editor adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the ANT transformer.

The Parameters Tab

The **Parameters** tab allows you to configure the parameters that are accessible as Ant properties in the Ant build script.

The table displays all the parameters that are available in the Ant build script, along with their description and current values. You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.



Note: To input parameter values at runtime, use the *ask editor variable* in the **Value** column.

The following actions are available for managing parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the  **Insert Editor Variables** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

These parameters are also available for the built-in validation processor and the **Content Completion Assistant**.

The Output Tab

The **Output** tab contains the following options:

- **Open** - Allows you to specify the file to open automatically when the transformation is finished. Usually, this is the output file of the Ant process. An *editor variable* can be inserted for the path by using the  **Insert Editor Variables** button.
 - **In System Application** - The file specified in the **Open** text box is opened in the system application that is set in the operating system as the default application for that type of file (for example, .pdf files are usually opened in the *Acrobat Reader* application).
 - **In Editor** - The file specified in the **Open** text box is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor).
- The **Show console output** option allows you to specify when to display the console output log. The following options are available:
 - **When build fails** - displays the console output log if the build fails.
 - **Always** - displays the console output log, regardless of whether or not the build fails.

XSLT Transformation

To create an **XSLT transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XSLT transformation**.
- Use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XSLT transformation**.
- Use the  **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XSLT transformation**.

 **Note:** If a scenario is already associated with the edited document, selecting  **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.

- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs:

- **XSLT**.
- **FO Processors**.
- **Output**.

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.
 -  **Note:** If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.
 -  **Note:** If the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and *the name of an initial template* is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty if you use *external XSLT processors*. Otherwise, a value is mandatory in the **XML URL** field.
- **XSL URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the **Data Source Explorer** window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens the file in an editor panel with the path that is specified in the **XML URL** text box.

The rest of the options available in the **XSLT** tab allow you to further customize the transformation scenario:

- **Use "xmlstylesheet" declaration** - Use the stylesheet declared with an `xmlstylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default, this checkbox is not selected and the transformation applies the XSLT stylesheet that is specified in the **XSL URL** field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xmlstylesheet` processing instruction.
- **Transformer** - This drop-down list presents all the transformation engines available to Oxygen XML Editor for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).

- **Advanced options** - Allows you to configure the advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the [Saxon-HE/PE/EE preferences page](#). For the current transformation scenario, these **Advanced options** override the options configured in the [Saxon-HE/PE/EE preferences page](#). The **Initial mode and template** option is only available in the **Advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template that starts the XSLT transformation or the initial mode of the transformation.
- **Parameters** - Opens [the Configure parameters dialog](#), allowing you to configure the XSLT parameters used in the current transformation. In this dialog you can also configure the parameters of additional stylesheets by using the **Additional XSLT stylesheets** button. If the XSLT transformation engine is custom-defined, you can not use this dialog to configure the parameters sent to the custom engine. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line that starts the transformation process.
- **Extensions** - Opens [the dialog for configuring the XSLT/XQuery extension jars or classes](#) that define extension Java functions or extension XSLT elements used in the transformation.
- **Additional XSLT stylesheets** - Opens [the dialog for adding XSLT stylesheets](#) that are applied on the main stylesheet that is specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XSLT result as input** - The FO processor is applied to the result of the XSLT transformation that is defined in the **XSLT** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an [external processor](#).

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include [special Oxygen XML Editor editor variables](#) or [custom editor variables](#) by using the  **Insert Editor Variables** button..
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the *Acrobat Reader* application).



Note: To set the web browser that is used for displaying HTML/XHTML pages, [open the Preferences dialog box](#), then go to **Global** and set it in the **Default Internet browser** field.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor opens the file specified here. The file path can include [special Oxygen XML Editor editor variables](#) or [custom editor variables](#) by using the  **Insert Editor Variables** button.
- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**

- **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.



Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **XML** - If this is checked, Oxygen XML Editor displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
- **SVG** - If this is checked, Oxygen XML Editor displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

XProc Transformation

A sequence of transformations described by an XProc script can be executed with an XProc transformation scenario. To create an **XProc transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XProc transformation**.
- Use the **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XProc transformation**.
- Use the **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XProc transformation**.



Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs:

- **XProc**
- **Inputs**
- **Parameters**
- **Outputs**
- **Options**

The XProc Tab

The **XProc** tab contains the following options:

- **XProc URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter value in the **XProc URL**:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

- **Processor** - Allows you to select the XProc engine. You can select the built-in *Calabash* engine or a custom engine that is *configured in the Preferences dialog*.

The Inputs Tab

The **Inputs** tab contains a list with the ports that the XProc script uses to read input data. Use the **Filter** text box to search for a specific term in the entire ports collection.

Each input port has an assigned name in the XProc script. The XProc engine reads data from the URL specified in the **URL** column. The *built-in editor variables* and *custom editor variables* can be used to specify the URL.

The following actions are available for managing the input ports:

New

Opens an **Edit** dialog that allows you to add a new port and its URL.

Edit

Opens an **Edit** dialog that allows you to modify the selected port and its URL.

Delete

Removes the selected port from the list. It is enabled only for new ports that have been added to the list.

The Parameters Tab

The **Parameters** tab presents a list of ports and parameters collected from the XProc script. The tab is divided into three sections:

- *List of Ports* - In this section you can use the **New** and **Delete** buttons to add or remove ports.
- *List of Parameters* - This section presents a list of parameters for each port and includes columns for the parameter name, namespace URI, and its value. Use the **Filter** text box to search for a specific term in the entire parameters collection. You can use the **New** and **Delete** buttons to add or remove parameters. You can edit the value of each cell in this table by double-clicking on the cell. You can also sort the parameters by clicking on the column headers.
- *Editor Variable Information* - The *built-in editor variables* and *custom editor variables* can be used for specifying the URI. The message pane at the bottom of the dialog provides more information about the editor variables that can be used.

The Outputs Tab

The **Outputs** tab displays a list of output ports (along with the URL) collected from the XProc script. Use the **Filter** text box to search for a specific term in the entire ports collection. You can also sort the columns by clicking on the column headers.

The following actions are available for managing the output ports:

New

Opens an **Edit** dialog that allows you to add a new output port and its URL. An *editor variable* can be inserted for the URL by using the  **Insert Editor Variables** button. There is also a **Show in transformation results view** option that allows you to select whether or not the results will be displayed in the output results view.

Edit

Opens an **Edit** dialog that allows you to edit an existing output port and its URL. An *editor variable* can be inserted for the URL by using the  **Insert Editor Variables** button. There is also a **Show in transformation results view** option that allows you to select whether or not the results will be displayed in the output results view.

Delete

Removes the selected output port from the list. It is enabled only for new ports that have been added to the list.

Additional options that are available at the bottom of this tab include:

Open in Editor

If this option is selected, the XProc transformation result is automatically opened in an editor panel.

Open in Browser/System Application

If this option is selected, you can specify a file to be opened at the end of the XProc transformation in the browser or system application that is associated with the file type. An *editor variable* can be inserted for the path by using the  **Insert Editor Variables** button.

Results

The result of the XProc transformation can be displayed as a sequence in an output view with two sections:

- A list with the output ports on the left side.
- The content that correspond to the selected output port on the right side.

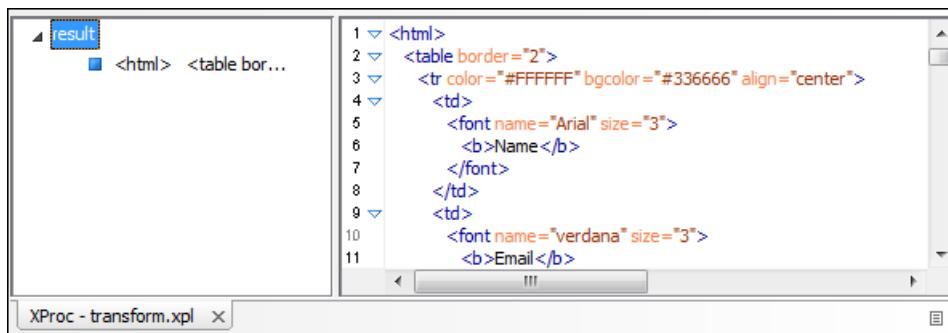


Figure 282: XProc Transformation Results View

The Options Tab

The **Options** tab displays a list of the options collected from the XProc script. The tab is divided into two sections:

- *List of Options* - This section presents a list of options and includes columns for the option name, namespace URI, and its value. Use the **Filter** text box to search for a specific term in the entire options collection. You can use the **New** and **Delete** buttons to add or remove options. You can edit the value of each cell in this table by double-clicking on the cell. You can also sort the parameters by clicking on the column headers. The names of edited options are displayed in bold.
- *Editor Variable Information* - The *built-in editor variables* and *custom editor variables* can be used for specifying the URI. This section provides more information about the editor variables that can be used.

Configuring Calabash with XEP

To generate PDF output from your XProc pipeline (when using the Calabash XProc processor), follow these steps:

1. Open the `[OXYGEN_DIR]/lib/xproc/calabash/engine.xml` file.
2. Uncomment the `<system-property name="com.xmlcalabash.fo-processor" value="com.xmlcalabash.util.FoXEP" />` system property.
3. Uncomment the `<system-property name="com.renderx.xep.CONFIG" file="../../../../tools/xep/xep.xml" />` system property. Edit the `file` attribute to point to the configuration file that is usually located in the XEP installation folder.
4. Uncomment the references to the XEP libraries. Edit them to point to the matching library names from the XEP installation directory.
5. Restart Oxygen XML Editor.

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download from the application website as a zip file [xprocAPI.zip](#). To create an XProc integration project, follow these steps:

1. Move the `oxygen.jar` file from `[OXYGEN_DIR]/lib` to the `lib` folder of your project.
2. Implement the `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface` interface.
3. Create a Java archive (jar) from the classes you created.
4. Create a `engine.xml` file according with the `engine.dtd` file. The attributes of the `engine` element are as follows:
 1. `name` - The name of the XProc engine.
 2. `description` - A short description of the XProc engine.
 3. `class` - The complete name of the class that implements `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface`.
 4. `version` - The version of the integration.
 5. `engineVersion` - The version of the integrated engine.
 6. `vendor` - The name of the vendor / implementer.
 7. `supportsValidation` - `true` if the engine supports validation, `false` otherwise.

The `engine` element has only one child, `runtime`. The `runtime` element contains several `library` elements with the `name` attribute containing the relative or absolute location of the libraries necessary to run this integration.

5. Create a folder with the name of the integration in the `[OXYGEN_DIR]/lib/xproc`.
6. Place the `engine.xml` and all the libraries necessary to run the new integration in that folder.

XQuery Transformation

To create an **XQuery transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XQuery transformation**.
- Use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XQuery transformation**.
- Use the  **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **XQuery transformation**.

 **Note:** If a scenario is already associated with the edited document, selecting  **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and **Storage** options:

- **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
- **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.

The lower part of the dialog box contains the following tabs:

- **XQuery**
- **FO Processor**
- **Output**

The XQuery Tab

The **XQuery** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.
-  **Note:** If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.
- **XQuery URL** - specifies the source XQuery file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens the file in an editor panel with the path that is specified in the **XML URL** text box.

The rest of the options available in the **XQuery** tab allow you to further customize the transformation scenario:

- **Transformer** - This drop-down list presents all the transformation engines available to Oxygen XML Editor for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).;
-  **Advanced options** - *configure advanced options specific for the Saxon HE / PE / EE engine*.

- **Parameters** - Opens the **Configure parameters** dialog for configuring the XQuery parameters. You can use buttons in this dialog you can add, edit, or remove parameters. If the XQuery transformation engine is custom-defined you can not use this dialog to set parameters. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XQuery engine to include the necessary parameters in the command line that starts the transformation process.



Note: Use the **Filter** text box to search for a specific term in the entire parameters collection.

- **Extensions** - Opens *the dialog for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XQuery result as input** - the FO processor is applied to the result of the XQuery transformation defined in the **XQuery** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Present as a sequence** - Enabling this option will reduce the time necessary to fetch the full result, as it will only fetch the first chunk of the result.
- **Prompt for file** - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the **Insert Editor Variables** button..
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the *Acrobat Reader* application).



Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the Preferences dialog box*, then go to **Global** and set it in the **Default Internet browser** field.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor opens the file specified here. The file path can include *special Oxygen XML Editor editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.
- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
 - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.



Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid

delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **XML** - If this is checked, Oxygen XML Editor displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
- **SVG** - If this is checked, Oxygen XML Editor displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

SQL Transformation

To create an **SQL transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **SQL transformation**.
- Use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **SQL transformation**.
- Use the  **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then click the **New** button and select **SQL transformation**.



Note: If a scenario is already associated with the edited document, selecting  **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the following options that control the transformation:

- **Name** - The unique name of the SQL transformation scenario.
- **Storage** - Allows you to select one of the following storage options:
 - **Global Options** - The scenario is saved in the global options that are stored in the user home directory and is not accessible to other users.
 - **Project Options** - The scenario is stored in the project file and can be shared with other users. For example, if your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) or a shared folder, your team can use the scenarios that you store in the project file.
- **SQL URL** - Allows you to specify the URL of the SQL script. You can use the following browsing buttons to enter value in this field:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the [Data Source Explorer](#) window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens the file in an editor panel with the path that is specified in the **XML URL** text box.

- **Connection** - Allows you to select a connection from a drop-down list. To configure a connection, use the  **Advanced options** button to open the [data source preferences page](#).
- **Parameters** - Allows you to configure the parameters of the transformation.

Configure Transformation Scenario(s) Dialog Box

You can use the **Configure Transformation Scenarios(s)** dialog box to manage both the [built-in transformation scenarios](#) and the ones you create.

To open this dialog box, use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu.

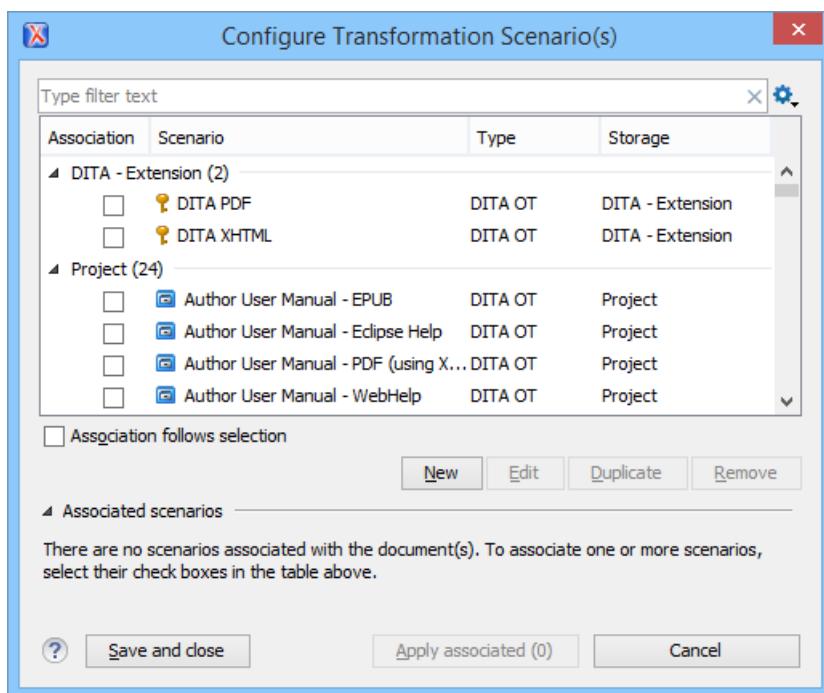


Figure 283: Configure Transformation Scenario(s) Dialog Box

The top section of the dialog box contains a filter that allows you to search through the scenarios list. The  **Settings** button allows you to configure the following options:

- **Show all scenarios** - Select this option to display all the available scenarios, regardless of the document they are associated with.
- **Show only the scenarios available for the editor** - Select this option to only display the scenarios that Oxygen XML Editor can execute for the current document type.
- **Show associated scenarios** - Select this option to only display the scenarios associated with the document you are editing.
-  **Import scenarios** - This option opens the **Import scenarios** dialog box that allows you to select the **scenarios** file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing

scenario, Oxygen XML Editor ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Editor adds `imported` to the name of the imported scenario.

- **Export selected scenarios** - Use this option to export transformation and validation scenarios individually. Oxygen XML Editor creates a `scenarios` file that contains the scenarios that you export.

The middle section of the dialog box displays the scenarios that you can apply to the current document. You can view both the scenarios associated with the current document type and the scenarios defined at project level. The following columns are used to display the transformation scenarios:

- **Association** - The check-boxes in this column mark whether a transformation scenario is associated with the current document.
- **Scenario** - This column presents the names of the transformation scenarios.
- **Type** - Displays the type of the transformation scenario. For further details about the different types of transformation scenarios available in Oxygen XML Editor see the [Defining a New Transformation Scenario](#) section.
- **Storage** - Displays where a transformation scenario is stored (the **Show Storage** option must be enabled.)

To sort each column you can left-click its header. The contextual menu of each header allows you to do the following:

- **Show Type** - Use this option to display the transformation type of each scenario.
- **Show Storage** - Use this option to display the storage location of the scenarios.
- **Group by Association** - Select this option to group the scenarios depending on whether or not they are associated with the current document.
- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.
- **Ungroup all** - Select this option to ungroup all the scenarios.
- **Reset Layout** - Select this option to restore the default settings of the layout.

The bottom section of the dialog box contains the following actions:

- **Association follows selection** - Enable this check-box to automatically associate selected transformation scenarios with the current document. This option can also be used for multiple selections.



Note: When this option is enabled, the **Association** column is hidden.

- **New** - This button allows you to create a new transformation scenario, *depending upon its type*.
- **Edit** - This button opens the **Edit Scenario** dialog box that allows you to configure the options of the transformations scenario.



Note: If you try to edit a transformation scenario associated with a defined document type, Oxygen XML Editor displays a warning message to inform you that this is not possible and gives you the option to create a *duplicate transformation scenario* to edit instead.

- **Duplicate** - Use this button to create a *duplicate transformation scenario*.
- **Remove** - Use this button to remove transformation scenarios.



Note: Removing scenarios associated with a defined document type is not allowed.

The **Edit**, **Duplicate**, **Remove**, **Import scenarios**, and **Export selected scenarios** actions are also available in the contextual menu of the transformation scenarios listed in the middle section of the dialog box.

This contextual menu also contains a **Change storage** action that allows you to change the storage location of a transformation scenario to **Global Options** or **Project Options**. You are also able to keep the original storage location and make a copy of the selected scenario in the new storage location.

Duplicating a Transformation Scenario

Use the following procedure to duplicate a transformation scenario. This is useful for creating a scenario that is similar to an existing one.

1. Open the **Configure Transformation** dialog by using the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu.
2. Create a copy of a scenario by selecting the scenario and clicking the **Duplicate** button.
3. Enter a new name in the **Name** field.
 - a) You can choose to save the scenarios at project level by selecting the **Project Options** setting.
4. Click **OK** to save the scenario.

Editing a Transformation Scenario

Editing a transformation scenario is useful if you need to configure some of its parameters.

Oxygen XML Editor allows you to configure existing transformation scenarios by using one of the following methods:

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Then select the scenario and click the  **Edit** button.
- Use the  **Configure Transformation Scenario(s) (Ctrl Shift C (Command Shift C on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then select the scenario and click the **Edit** button.
- Use the  **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu. Then select the scenario and click the **Edit** button.



Note: If a scenario is already associated with the edited document, selecting  **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

You can edit transformation scenarios that are defined at project level only. To edit a transformation scenario that is associated with a defined document type, duplicate it and edit the duplicated scenario.

Apply Batch Transformations

A transformation action can be applied on a batch of selected files *from the contextual menu of the Project view* without having to open the files involved in the transformation. You can apply the same scenario to a batch of files or multiple scenarios to a single file or batch of files.

1. (Optional, but recommended) Organize the files you want to transform in logical folders.
 - a) Create a logical folder in the **Project** view by using the **New > Logical Folder...** action from the contextual menu of the root file.
 - b) Add files you want to transform to the logical folder by using the  **Add Files...** or  **Add Edited File** actions from the contextual menu of the logical folder.



Note: You can skip this step if the files are already in a dedicated folder that does not include any additional files or folders. You can also manually select the individual files in the **Project** view each time you want to transform them, but this can be tedious.

2. Right-click on the newly created logical folder and select **Transform > Configure Transformation Scenario(s)...** to select one or more transformation scenarios to be applied on all the files in the logical folder.

Note: These types of transformation scenarios must be configured with the current file (`${cf}`) or current file URL (`${currentFileURL}`) editor variables for the input file. This ensures that each file becomes the current file when its turn arrives in the batch transformation process. Edit the transformation scenario to make sure the appropriate editor variable is assigned for the input file. For example, for a DocBook PDF transformation make sure the **XML URL** input box is set to the `${currentFileURL}` editor variable. For a DITA PDF transformation make sure the `args.input` parameter is set to the `${cf}` editor variable.

3. Now that logical folder has been associated with one or more transformation scenarios, whenever you want to apply the same batch transformation you can select **Transform > Transform with...** from the contextual menu and the same previously associated scenario(s) will be applied.
4. If you want a different type of transformation to be applied to each file inside the logical folder, associate individual scenarios for each file and select **Transform > Apply Transformation Scenario(s)** from the contextual menu of the logical folder.

Built-in Transformation Scenarios

Oxygen XML Editor included preconfigured built-in transformation scenarios that are used for common transformations.

To obtain the desired output, use the **Apply Transformation Scenario(s) (Ctrl Shift T (Command Shift T on OS X))** action from the **Transformation** toolbar or the **Document > Transformation** menu and choose one of the built-in scenarios for the current document.

You can use the **Apply Transformation Scenario(s)** action even if the current document is not associated with a transformation scenario.

If the document contains an `xmlstylesheet` processing instruction that refers to an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Editor prompts you to associate the document with a built-in transformation scenario.

The default transformation scenario is suggested based on the processing instruction from the edited document. The **XSL URL** field of the default transformation scenario contains the URL from the `href` attribute of the processing instruction. By default, the **Use xmlstylesheet declaration** check-box is enabled, Saxon is used as the transformation engine, and no FO processing is performed. The result of the transformation is stored in a file with the same URL as the edited document, but the extension is changed to `html`. The name and path are preserved because the output file name is specified with the help of two *editor variables*: `${cf}` and `${fn}` .

Sharing the Transformation Scenarios

The transformation scenarios can be shared with other users by saving them at project level. When you create a new transformation scenario or edit an existing one, there is a **Storage** option to control whether the scenarios are stored in **Global Options** or **Project Options**.

Storage: Global Options Project Options

Selecting **Global Options** stores the scenario in the global options that are stored in the user home directory.

Selecting **Project Options** stores the scenario in the project file and can be shared with other users that have access to the project. If your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc.) then your team will have access to the scenarios that you define. When you create a scenario at the project level, the URLs from the scenario become relative to the project URL.

When using the **Apply Transformation Scenario(s)** or **Configure Transformation Scenario(s)** actions, the predefined scenarios are presented according to the current detected document type, along with your created custom scenarios. The following screenshot shows default scenarios for a *DITA* document along with several custom

transformation scenarios. The  key symbol before the scenario name indicates that the scenario can only be modified from the [Document Type Association](#) preferences page.

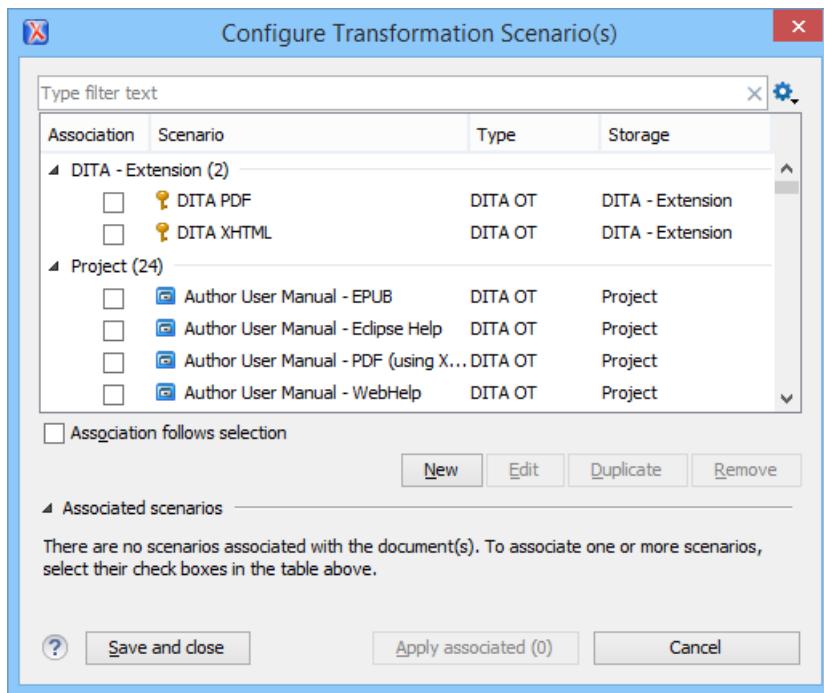


Figure 284: Transformation Scenario List Dialog

You can also change the storage options on existing transformation scenarios by using the **Change storage** action from the contextual menu of the list of scenarios.

Other preferences can also be stored at the project level. For more information, see the [Preference Sharing](#) section.

Transformation Scenarios View

You can manage the transformation scenarios by using the **Transformation Scenarios** view. To open this view, go to **Window > Show View > Transformation Scenarios**.

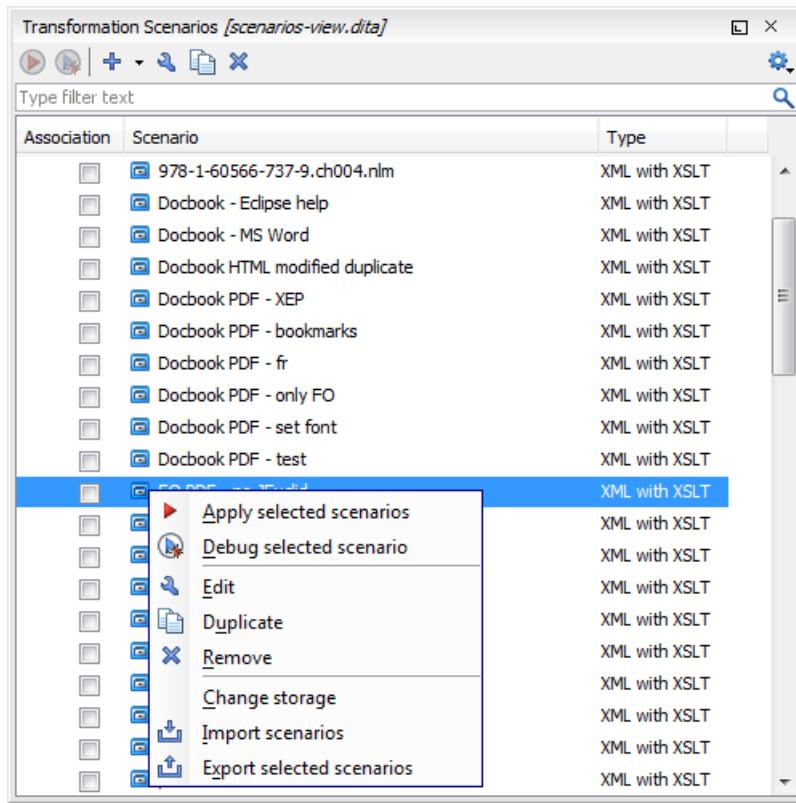


Figure 285: The Scenarios view

The following options are available in the contextual menu of the **Transformation Scenarios** view:

Apply selected scenarios

Select this option to run the current transformation scenario.

Debug selected scenario

Select this option to switch to the **Debugger** perspective and initialize it with the parameters from the scenario (the XML, XSLT, or XQuery input, the transformation engine, the XSLT parameters).

Duplicate

Adds a new scenario to the list that is a duplicate of the current scenario. It is useful for creating a scenario that is similar to an existing one.

Edit

Opens the dialog for editing the parameters of a transformation scenario.

Remove

Removes the current scenario from the list. This action is also available by using the **Delete** key.

Change storage

Use this option to change the storage location of the selected scenario. You are also able to keep the original storage location and make a copy of the selected scenario in the target storage location.

Import scenarios

This option opens the **Import scenarios** dialog that allows you to select the **scenarios** file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Editor ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.

- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Editor adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export transformation and validation scenarios individually. Oxygen XML Editor creates a `scenarios` file that contains the scenarios that you export.

Along with the options available in the contextual menu, the **Transformation Scenarios** view toolbar contains a

- + - New drop-down button that contains a list of the scenarios you can create. Oxygen XML Editor determines the most appropriate scenarios for the current type of file and displays them at the beginning of the list, followed by the rest of the scenarios.

The Settings drop-down menu allows you to configure the following options:

- **Show all scenarios** - Select this option to display all the available scenarios, regardless of the document they are associated with.
- **Show only the scenarios available for the editor** - Select this option to only display the scenarios that Oxygen XML Editor can execute for the current document type.
- **Show associated scenarios** - Select this option to only display the scenarios associated with the document you are editing.
- **Change storage** - Use this option to change the storage location of the selected scenario to **Global Options** or **Project Options**. You are also able to keep the original storage location and make a copy of the selected scenario in the new storage location.
- **Import scenarios** - This option opens the **Import scenarios** dialog box that allows you to select the `scenarios` file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Editor ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:
 - Keep or replace the existing scenario.
 - Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Editor adds `imported` to the name of the imported scenario.

- **Export selected scenarios** - Use this option to export transformation and validation scenarios individually. Oxygen XML Editor creates a `scenarios` file that contains the scenarios that you export.
- **Show Type** - Use this option to display the transformation type of each scenario.
- **Show Storage** - Use this option to display the storage location of the scenarios.
- **Group by Association** - Select this option to group the scenarios depending on whether or not they are associated with the current document.
- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.
- **Ungroup all** - Select this option to ungroup all the scenarios.
- **Reset Layout** - Select this option to restore the default settings of the layout.

Oxygen XML Editor supports multiple scenarios association. To associate multiple scenarios with a document, enable the check-boxes in front of each scenario. You can also associate multiple scenarios with a document from the **Configure Transformation Scenario(s)** or **Configure Validation Scenario(s)** dialogs.

The **Transformation Scenarios** presents both global scenarios and project scenarios. By default, Oxygen XML Editor presents the items in the **Transformation Scenarios** in the following order: scenarios matching the current framework, scenarios matching the current project, scenarios matching other frameworks. You can group the scenarios depending

on the columns in the **Transformation Scenarios** view. Right click the name of a column to choose how to group the scenarios. The following grouping options are available:

- **Group by Association** - Select this option to group the scenarios depending on whether or not they are associated with the current document.
- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.

Debugging PDF Transformations

To debug a DITA PDF transformation scenario using the XSLT Debugger follow these steps:

1. *Open the Preferences dialog box*, go to **XML > XML Catalog**, click **Add**, and select the file located at `[OXYGEN_DIR]\frameworks\dita\{DITA-OT}\plugins\org.dita.pdf2\cfg\catalog.xml`.
2. Open the map in the **DITA Maps Manager** and create a **DITA Map PDF** transformation scenario.
3. Edit the scenario, go to the **Parameters** tab and change the value of the **clean.temp** parameter to **no**.
4. Run the transformation scenario.
5. Open the **stage1.xml** file located in the temporary directory and *format and indent* it.
6. Create a transformation scenario for this XML file by associating the `topic2fo_shell_fop.xsl` stylesheet located at `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl`. If you are specifically using the RenderX XEP or Antenna House FO processors to build the PDF output, you should use the XSL stylesheets `topic2fo_shell_xep.xsl` or `topic2fo_shell_axf.xsl` located in the same folder.
7. In the transformation scenario edit the XSLT Processor combo box choose the Saxon EE XSLT processor (the same processor used when the DITA OT transformation is executed).
8. In the transformation scenario edit the **Parameters** list and set the parameter `locale` with the value `en_GB` and the parameter `customizationDir:url` to point either to your customization directory or to the default DITA OT customization directory. Its value should have an URL syntax like:`file:///c:/path/to/[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg`.
9. Debug the transformation scenario.

XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Editor.

Supported XSLT Processors

Oxygen XML Editor includes the following XSLT processors:

- **Xalan 2.7.1** - *Xalan-Java* is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- **Saxon 6.5.5** - *Saxon 6.5.5* is an XSLT processor that implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.
- **Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE)** - *Saxon-HE/PE* implements the basic conformance level for XSLT 2.0 / 3.0 and XQuery 1.0. The term *basic XSLT 2.0 / 3.0 processor* is defined in the draft XSLT 2.0 / 3.0 specifications. It is a conformance level that requires support for all features of the language other than those that involve schema processing. The HE product remains open source, but removes some of the more advanced features that are present in Saxon-PE.
- **Saxon 9.6.0.5 Enterprise Edition (EE)** - *Saxon EE* is the schema-aware edition of Saxon and it is one of the built-in processors included in Oxygen XML Editor. Saxon EE includes an XML Schema processor, and schema-aware XSLT, XQuery, and XPath processors.

The validation in schema aware transformations is done according to the W3C XML Schema 1.0 or 1.1. This can be [configured in Preferences](#).

-  **Note:** Oxygen XML Editor implements a Saxon framework that allows you to create Saxon configuration files. Two templates are available: **Saxon collection catalog** and **Saxon configuration**. Both of these templates support content completion, element annotation, and attribute annotation.

-  **Note:** Saxon can use the *ICU-J localization library* (`saxon9-icu.jar`) to add support for sorting and date/number formatting in a wide variety of languages. This library is not included in the Oxygen XML Editor installation kit. However, Saxon will use the default collation and localization support available in the currently used JRE. To enable this capability follow these steps:

1. Download Saxon 9.6.0.5 Professional Edition (PE) or Enterprise Edition (EE) from <http://www.saxonica.com>.
2. Unpack the downloaded archive.
3. Copy the `saxon9-icu.jar` file to `oxygen/lib` directory.
4. Re-start the application.

- **Saxon-CE (Client Edition)** is Saxonica's implementation of XSLT 2.0 for use on web browsers. Oxygen XML Editor provides support for editing stylesheets that contain Saxon-CE extension functions and instructions. This support improves the validation, content completion, and syntax highlighting.

-  **Note:** Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Editor only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about [executing such a stylesheet on Saxonica's website](#).

-  **Note:** A specific template, named **Saxon-CE stylesheet**, is available in the [New Document wizard](#).

- **Xsltproc (libxslt)** - *Libxslt* is the XSLT C library developed for the Gnome project. *Libxslt* is based on *libxml2*, the XML C library developed for the Gnome project. It also implements most of the EXSLT set of processor-portable extensions, functions, and some of Saxon's evaluate and expression extensions. The *libxml2* version included in Oxygen XML Editor is 2.7.6 and the *Libxslt* version is 1.1.26.

Oxygen XML Editor uses *Libxslt* through its command line tool (`Xsltproc`). The XSLT processor is included in the distribution kit of the stand-alone version for Windows and Mac OS X. Since there are differences between various Linux distributions, on Linux you must install *Libxslt* on your machine as a separate application and set the PATH variable to contain the `Xsltproc` executable.

-  **Note:** The `Xsltproc` processor can be configured from the [XSLTPROC options page](#).

-  **Caution:** Known problem: file paths containing spaces are not handled correctly in the LIBXML processor. For example, the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled properly by LIBXML if Oxygen XML Editor is installed in the default location on Windows (C:\Program Files). This is because the built-in XML catalog files are stored in the `[OXYGEN_DIR]/frameworks` subdirectory of the installation directory, which in this case contains at least a space character.

- **MSXML 4.0** - *MSXML 4.0* is available only on Windows platforms. It can be used for [transformation](#) and [validation of XSLT stylesheets](#).

Oxygen XML Editor uses the Microsoft XML parser through its command line tool `msxsl.exe`.

Since `msxsl.exe` is only a wrapper, Microsoft Core XML Services (MSXML) must be installed on the computer. Otherwise, you will get a corresponding warning. You can get the latest Microsoft XML parser from [Microsoft web-site](#).

- **MSXML .NET** - *MSXML .NET* is available only on Windows platforms. It can be used for [transformation](#) and [validation of XSLT stylesheets](#).

Oxygen XML Editor performs XSLT transformations and validations using .NET Framework's XSLT implementation (`System.Xml.Xsl.XslTransform` class) through the `nxslt` command line utility. The `nxslt` version included in Oxygen XML Editor is 1.6.

You should have the .NET Framework version 1.0 already installed on your system. Otherwise, you will get the following warning: `MSXML.NET requires .NET Framework version 1.0 to be installed.`
`Exit code: 128.`

You can get the .NET Framework version 1.0 from the [Microsoft website](#).

- **.NET 1.0** - A transformer based on the `System.Xml` 1.0 library available in the .NET 1.0 and .NET 1.1 frameworks from Microsoft (<http://msdn.microsoft.com/xml/>). It is available only on Windows.

You should have the .NET Framework version 1.0 or 1.1 already installed on your system. Otherwise, you will get the following warning: `MSXML.NET requires .NET Framework version 1.0 to be installed.`
`Exit code: 128.`

You can get the .NET Framework version 1.0 from the [Microsoft website](#).

- **.NET 2.0** - A transformer based on the `System.Xml` 2.0 library available in the .NET 2.0 framework from [Microsoft](#). It is available only on Windows.

You should have the .NET Framework version 2.0 already installed on your system. Otherwise, you will get the following warning: `MSXML.NET requires .NET Framework version 2.0 to be installed.`
`Exit code: 128.`

You can get the .NET Framework version 2.0 from the [Microsoft website](#).

For information about configuring the XSLT preferences, see the [XSLT options](#) section.

Configuring Custom XSLT Processors

You can configure and run XSLT and XQuery transformations with processors other than [the ones which come with the Oxygen XML Editor distribution](#).



Note: You can not use these custom engines in [the Debugger perspective](#).

The output messages of a custom processor are displayed in an output view at the bottom of the application window. If an output message follows [the format of an Oxygen XML Editor linked message](#), then a click on the message in the output view highlights the location of the message in an editor panel containing the file referenced in the message.

Configuring the XSLT Processor Extensions Paths

The Xalan and Saxon processors support the use of extension elements and extension functions. Unlike a literal result element, which the stylesheet simply transfers to the result tree, an extension element performs an action. The extension is usually used because the XSLT stylesheet fails in providing adequate functions to the user for accomplishing a more complex task.

The DocBook extensions for Xalan and Saxon are included in the
`[OXYGEN_DIR]\frameworks\docbook\xsl\extensions` folder.

Samples on how to use extensions can be found at:

- for Xalan - <http://xml.apache.org/xalan-j/extensions.html>
- for Saxon 6.5.5 - <http://saxon.sourceforge.net/saxon6.5.5/extensions.html>
- for Saxon 9.6.0.5 - <http://www.saxonica.com/documentation9.5/index.html#!extensibility>

To set an XSLT processor extension (a directory or a `.jar` file), use [the Extensions button](#) in the **Edit scenario** dialog box.



Note: The old way of setting an extension (using the parameter `-Dcom.oxygenxml.additional.classpath`) was deprecated, and instead you should use the extension mechanism of the XSLT transformation scenario.

XSL-FO Processors

This section explains how to apply XSL-FO processors when transforming XML documents to various output formats in Oxygen XML Editor.

The Built-in XSL-FO Processor

The Oxygen XML Editor installation package is distributed with the *Apache FOP* that is a Formatting Objects processor for rendering your XML documents to PDF. *FOP* is a print and output independent formatter driven by XSL Formatting Objects. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

To include PNG images in the final PDF document you need the *JIMI* or *JAI* libraries. For PDF images you need the *fop-pdf-images* library. These libraries are not bundled with Oxygen XML Editor but using them is very easy. You need to download them and *create an external FO processor* based on the built-in FOP libraries and the extension library. The *external FO processor created in Preferences* will have a command line like:

```
java -cp "${oxygenInstallDir}/lib/xercesImpl.jar:  
${oxygenInstallDir}/lib/fop.jar:${oxygenInstallDir}/lib/  
avalon-framework-4.2.0.jar:  
${oxygenInstallDir}/lib/batik-all-1.7.jar:${oxygenInstallDir}/lib/  
commons-io-1.3.1.jar:  
${oxygenInstallDir}/lib/xmlgraphics-commons-1.3.1.jar:  
${oxygenInstallDir}/lib/commons-logging-1.0.4.jar:  
${oxygenInstallDir}/lib/saxon9ee.jar:${oxygenInstallDir}/lib/  
saxon9-dom.jar:  
${oxygenInstallDir}/lib/xalan.jar:${oxygenInstallDir}/lib/  
serializer.jar:  
${oxygenInstallDir}/lib/resolver.jar:${oxygenInstallDir}/lib/  
fop-pdf-images-1.3.jar:  
${oxygenInstallDir}/lib/PDFBox-0.7.3.jar"  
org.apache.fop.cli.Main -fo ${fo} -${method} ${out}
```

You need to add to the classpath *JimiProClasses.zip* for *JIMI* and *jai_core.jar*, *jai_codec.jar* and *mlibwrapper_jai.jar* for *JAI*. For the *JAI* package you can include the directory containing the native libraries (*mlib_jai.dll* and *mlib_jai_mm.dll* on Windows) in the *PATH* system variable.

The OS X version of the *JAI* library can be downloaded from <http://www.apple.com/downloads/macosx/apple/java3dandjavaadvancedimagingupdate.html>. In order to use it, install the downloaded package.

Other FO processors can be configured in *the Preferences dialog box*.

Add a Font to the Built-in FOP - The Simple Version

If the font that must be set to Apache FOP is one of the fonts that are installed in the operating system you should follow the next steps for creating and setting a FOP configuration file that looks for the font that it needs in the system fonts. It is a simplified version of *the procedure for setting a custom font in Apache FOP*.

1. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)
 - a) Create a FOP configuration file that specifies that FOP should look for fonts in the installed fonts of the operating system.

```
<fop version="1.0">  
  <renderers>  
    <renderer mime="application/pdf">  
      <font>  
        <auto-detect/>  
      </font>  
    </renderer>  
  </renderers>  
</fop>
```

- b) *Open the Preferences dialog box*, go to **XML > XSLT/FO/XQuery > FO Processors**, and enter the path of the FOP configuration file in the **Configuration file** text field.
2. Set the font on the document content.

This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

- For DocBook documents you can start with the predefined scenario called **DocBook PDF**, [edit the XSLT parameters](#) and set the font name (in our example the font family name is **Arial Unicode MS**) to the parameters body.font.family and title.font.family.
- For TEI documents you can start with the predefined scenario called **TEI PDF**, [edit the XSLT parameters](#) and set the font name (in our example **Arial Unicode MS**) to the parameters bodyFont and sansFont.
- For DITA transformations to PDF using DITA-OT you should modify the following two files:
 - [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml
- the font-face element included in each element physical-font having the attribute char-set="default" must contain the name of the font (**Arial Unicode MS** in our example)
 - [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf
- an element auto-detect must be inserted in the element fonts which is inside the element renderer having the attribute mime="application/pdf":

```
<renderer mime="application/pdf">
  .
  .
  <font>
    <auto-detect/>
  </font>
  .
</renderer>
```

Add a Font to the Built-in FOP

If an XML document is transformed to PDF using the built-in Apache FOP processor but it contains some Unicode characters that cannot be rendered by the default PDF fonts, then a special font that is capable to render these characters must be configured and embedded in the PDF result.

 **Important:** If this special font is installed in the operating system, there is a simple way of telling FOP to look for it. See [the simplified procedure for adding a font to FOP](#).

1. Locate the font.

First, find out the name of a font that has the glyphs for the special characters you used. One font that covers most characters, including Japanese, Cyrillic, and Greek, is Arial Unicode MS.

On Windows the fonts are located into the C:\Windows\Fonts directory. On Mac, they are placed in /Library/Fonts. To install a new font on your system, is enough to copy it in the Fonts directory.

2. Generate a font metrics file from the font file.

- Open a terminal.
- Change the working directory to the Oxygen XML Editor install directory.
- Create the following script file in the Oxygen XML Editor installation directory.

For OS X and Linux create a file ttfConvert.sh:

```
#!/bin/sh

export LIB=lib
export CP=$LIB/fop.jar
export CP=$CP:$LIB/avalon-framework-4.2.0.jar
export CP=$CP:$LIB/xercesImpl.jar
export CP=$CP:$LIB/commons-logging-1.1.1.jar
export CP=$CP:$LIB/commons-io-1.3.1.jar
export CP=$CP:$LIB/xmlgraphics-commons-1.5.jar
export CP=$CP:$LIB/xml-apis.jar
export CMD="java -cp $CP org.apache.fop.apps.TTFFReader"
export FONT_DIR='.'

$CMD $FONT_DIR/Arialuni.ttf Arialuni.xml
```

For Windows create a file ttfConvert.bat:

```
@echo off
set LIB=lib
set CP=%LIB%\fop.jar
```

```

set CP=%CP%;%LIB%\avalon-framework-4.2.0.jar
set CP=%CP%;%LIB%\xercesImpl.jar
set CP=%CP%;%LIB%\commons-logging-1.1.1.jar
set CP=%CP%;%LIB%\commons-io-1.3.1.jar
set CP=%CP%;%LIB%\xmlgraphics-commons-1.5.jar
set CP=%CP%;%LIB%\xml-apis.jar
set CMD=java -cp "%CP%" org.apache.fop.fonts.apps.TTFFReader
set FONT_DIR=C:\Windows\Fonts
%CMD% %FONT_DIR%\Arialuni.ttf Arialuni.xml

```

The paths specified in the file are relative to the Oxygen XML Editor installation directory. If you decide to create it in other directory, change the file paths accordingly.

The *FONT_DIR* can be different on your system. Check that it points to the correct font directory. If the Java executable is not in the *PATH*, specify the full path of the executable.

If the font has bold and italic variants, convert them too by adding two more lines to the script file:

- for OS X and Linux:

```

$CMD $FONT_DIR/Arialuni-Bold.ttf Arialuni-Bold.xml
$CMD $FONT_DIR/Arialuni-Italic.ttf Arialuni-Italic.xml

```

- for Windows:

```

%CMD% %FONT_DIR%\Arialuni-Bold.ttf Arialuni-Bold.xml
%CMD% %FONT_DIR%\Arialuni-Italic.ttf Arialuni-Italic.xml

```

d) Execute the script.

On Linux and OS X, execute the command `sh ttfConvert.sh` from the command line. On Windows, run the command `ttfConvert.bat` from the command line or double click on the file `ttfConvert.bat`.

3. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)

a) Create a FOP configuration file that specifies the font metrics file for your font.

```

<fop version="1.0">
  <base>./</base>
  <font-base>file:/C:/path/to/FOP/font/metrics/files/</font-base>
  <source-resolution>72</source-resolution>
  <target-resolution>72</target-resolution>
  <default-page-settings height="11in" width="8.26in"/>
  <renderers>
    <renderer mime="application/pdf">
      <filterList>
        <value>flate</value>
      </filterList>
      <fonts>
        <font metrics-url="Arialuni.xml" kerning="yes"
              embed-url="file:/Library/Fonts/Arialuni.ttf">
          <font-triplet name="Arialuni" style="normal"
                        weight="normal"/>
        </font>
      </fonts>
    </renderer>
  </renderers>
</fop>

```

The `embed-url` attribute points to the font file to be embedded. Specify it using the URL convention. The `metrics-url` attribute points to the font metrics file with a path relative to the `base` element. The triplet refers to the unique combination of name, weight, and style (italic) for each variation of the font. In our case is just one triplet, but if the font had variants, you would have to specify one for each variant. Here is an example for Arial Unicode if it had italic and bold variants:

```

<fop version="1.0">
  ...
  <font-base>file:/Library/Fonts/</font-base>
  <font metrics-url="Arialuni.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni.ttf">
    <font-triplet name="Arialuni" style="normal"
                  weight="normal"/>
  </font>
  <font metrics-url="Arialuni-Bold.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni-Bold.ttf">
    <font-triplet name="Arialuni" style="bold"
                  weight="bold"/>
  </font>
  <font metrics-url="Arialuni-Italic.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni-Italic.ttf">
    <font-triplet name="Arialuni" style="italic"
                  weight="normal"/>
  </font>
</fop>

```

```

        <embed-url="file:/Library/Fonts/Arialuni-Bold.ttf">
        <font-triplet name="Arialuni" style="normal"
                      weight="bold"/>
    </font>
    <font metrics-url="Arialuni-Italic.xml" kerning="yes"
          embed-url="file:/Library/Fonts/Arialuni-Italic.ttf">
        <font-triplet name="Arialuni" style="italic"
                      weight="normal"/>
    </font>
</fonts>
...
</fop>
```

More details about the FOP configuration file are available on the [FOP website](#).

- b) [Open the Preferences dialog box](#), go to **XML > XSLT/FO/XQuery > FO Processors**, and enter the path of the FOP configuration file in the **Configuration file** text field.

4. Set the font on the document content.

This is usually done with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

For DocBook documents, you can start with the predefined scenario called **DocBook PDF**, [edit the XSLT parameters](#), and set the font name (in our example **Arialuni**) to the parameters `body.font.family` and `title.font.family`.

For TEI documents, you can start with the predefined scenario called **TEI PDF**, [edit the XSLT parameters](#), and set the font name (in our example **Arialuni**) to the parameters `bodyFont` and `sansFont`.

For DITA to PDF transformations using DITA-OT modify the following two files:

- [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml
- the `font-face` element included in each element `physical-font` having the attribute `char-set="default"` must contain the name of the font (*Arialuni* in our example)
- [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf
- an element `font` must be inserted in the element `fonts` which is inside the element `renderer` having the attribute `mime="application/pdf"`:

```

<renderer mime="application/pdf">
    <fonts>
        <font metrics-url="Arialuni.xml" kerning="yes"
              embed-url="file:/Library/Fonts/Arialuni.ttf">
            <font-triplet name="Arialuni" style="normal"
                          weight="normal"/>
        </font>
    </fonts>
</renderer>
```

Adding Libraries to the Built-in FOP

You can extend the functionality of the built-in FO processor by dropping additional libraries in the [OXYGEN_DIR]/lib/fop directory.

Hyphenation

To add support for hyphenation:

1. download the pre-compiled JAR from [OFFO](#) ;
2. place the JAR in [OXYGEN_DIR]/lib/fop;
3. restart the Oxygen XML Editor.

Output Formats

Oxygen XML Editor allows you to use transformation scenarios to publish XML content in various output formats (such as WebHelp, PDF, CHM, EPUB, JavaHelp, Eclipse Help, XHTML, etc.)

For transformations that are not included in your installed version of Oxygen XML Editor, simply install the tool chain required to perform the specific transformation and process the files in accordance with the processor instructions. A

multitude of target formats are possible. The basic condition for transformation to any format is that your source document is well-formed.



Note: You need to use the appropriate stylesheet according to the source definition and the desired output. For example, if you want to transform into an HTML format using a DocBook stylesheet, your source XML document should conform with the DocBook DTD.

For more information, see the [Transformation Scenarios](#) on page 690 section.

WebHelp Output

Oxygen XML Editor allows you to obtain WebHelp output from DocBook and DITA documents. This section contains information about the WebHelp system, its variants, and ways to customize it to better fit your specific needs.

WebHelp System Description

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Layout

The layout of the WebHelp system is comprised of two parts:

- The left section that contains separate tabs for **Content**, **Search**, and **Index**.



Note: If your documents contain no `indexterm` elements, the **Index** tab is not generated.



Note: You can enhance the appearance of the selected item in the Table of Contents. See the [Customizing WebHelp chapter](#) for more details.

- The right section where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper-right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.



Note: You can edit the `args.hide.parent.link` parameter to hide the **Parent**, **Next**, and **Previous** links.

You can use the **Collapse all** button that is displayed in the **Content** tab to collapse all the topics presented in the Table of Contents.

The top-right corner of the page contains the following options:

- **With Frames** - Displays the output using HTML frames to render two separate sections (a section that displays the Table of Contents in the left side and a section that displays the content of a topic in the right side).
- **Print this page** - Opens a dialog with various printing options and a print preview.

Growing Flowers

Content Search Index Flowers by Season / Summer Flowers

Lilac

From Wikipedia, the free encyclopedia.

Lilac (*Syringa*) is a [genus](#) of about 20–25 species of flowering plants in the olive family (*Oleaceae*), native to Europe and Asia.



They are deciduous shrubs or small trees, ranging in size from 2–10 m tall, with stems up to 20–30 cm diameter. The leaves are opposite (occasionally in whorls of three) in arrangement, and their shape is simple and heart-shaped to broad lanceolate in most species, but pinnate in a few species (e.g. *S. protolaciniata*, *S. pinnatifolia*). The flowers are produced in spring, each flower being 5–10 mm in diameter with a four-lobed corolla, the corolla tube narrow, 5–20 mm long. The usual flower colour is a shade of purple (often a light purple or lilac), but white and pale pink are also found. The flowers grow in large [panicle](#), and in several species have a strong fragrance. Flowering varies between mid spring to early summer, depending on the species. The fruit is a dry, brown capsule, splitting in two at maturity to release the two winged seeds.

Related information

[Gardenia](#)

WebHelp output generated by  XML Author.

Figure 286: WebHelp Output

Search Tab

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on the following:

- The number of keywords found in a single page (the higher the number, the better).
- The context (for example, a word found in a title scores better than a word found in unformatted text). The search ranking order, sorted by relevance is as follows:
 - The search phrase is included in a meta keyword
 - The search phrase is in the title of the page
 - The search phrase is in bold text in a paragraph
 - The search phrase is in normal text in a paragraph

Growing Flowers

Content Search Index Flowers by Season / Summer Flowers

Keywords: grow flowers

Search

Results for: flowers, grow

Lilac
From Wikipedia, the free encyclopedia. Lilac (*Syringa*) flowers summer lilac is a genus of about 20–25 species of flowering plants in the olive family (Oleaceae), native to Europe and Asia. They... 

Iris
From Wikipedia, the free encyclopedia. Iris flowers spring iris is a genus of... Rubiaceae, native to the tropical and subtropical... 

Lilac

From Wikipedia, the free encyclopedia.

Lilac (*Syringa*) is a [genus](#) of about 20–25 species of [flowering](#) plants in the olive family ([Oleaceae](#)), native to Europe and Asia.



They are deciduous shrubs or small trees, ranging in size from 2–10 m tall, with stems up to 20–30 cm diameter. The leaves are opposite (occasionally in whorls of three) in arrangement, and their shape is simple and heart-shaped to broad lanceolate in most species, but pinnate in a few species (e.g. *S. protolaciniata*, *S. pinnatifolia*). The [flowers](#) are produced in spring, each flower being 5–10 mm in diameter with a four-lobed corolla, the corolla tube narrow, 5–20 mm long. The usual [flower](#) colour is a shade of purple (often a light purple or lilac), but white and pale pink are also found. The [flowers](#) grow in large [panicle](#), and in several species have a strong fragrance. Flowering varies between mid spring to early summer, depending on the species. The fruit is a dry, brown capsule, splitting in two at maturity to release the two winged seeds.

Related information
[Gardenia](#)

WebHelp output generated by  XML Author.

Figure 287: WebHelp Search with Stemming Enabled

Rules that are applied during a search include:

- The space character separates keywords (an expression such as *grow flowers* counts as two separate keywords: *grow* and *flowers*).
- Do not use quotes to perform an exact search for multiple word expressions (an expression such as "*grow flowers*", returns no results since it searches for two separate words).
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page (for example, content inside `keywords` elements weighs twice as much as content inside an `H1` HTML element).
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters count as a single word.
- Always search for words containing three or more characters (shorter words, such as *to* or *of* are ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

This output format is compatible with the following browsers:

- Internet Explorer (8 or newer)
- Chrome
- Firefox
- Safari
- Opera

 **Important:** Due to some security restrictions in Google Chrome, WebHelp pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend that you load WebHelp pages in Google Chrome only from a web server (with a URL such as

<http://your.server.com/webhelp/index.html> or
http://localhost/web_pages/index.html.



Warning: Due to some restrictions in web browsers in regards to JavaScript code, the *frameless* version (index.html start page) of the WebHelp system should only be loaded from a web server (with a URL such as <http://your.server.com/webhelp/index.html> or http://localhost/web_pages/index.html). When loading WebHelp pages from the local file system, the *frameset* version (index_frames.html start page) of the WebHelp system should be used instead (file:///...).

WebHelp with Feedback System Description

WebHelp with Feedback is a form of online help system that consists of a series of web pages (XHTML format). Its advantages include platform independence, continuous content update, and a feedback mechanism that allows your authors and audience to interact with one another.

Layout

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp and the left section is the same. However, the bottom of the right section contains a **comments** bar. Select **Log in** from this bar to authenticate as a user for the WebHelp system. If you do not have a user name, complete the fields in the dialog box to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment, regardless of whether or not you are logged in.



Note: You can enhance the appearance of the selected item in the Table of Contents. See the [Customizing WebHelp chapter](#) for more details.

The screenshot shows the 'Growing Flowers' WebHelp system. The left sidebar contains a hierarchical Table of Contents (TOC) with categories like 'Introduction', 'Care and Preparation', 'Flowers by Season' (which is expanded to show 'Winter Flowers', 'Spring Flowers', 'Summer Flower', 'Autumn Flower', and 'Winter Flower'), 'Glossary', and 'Copyright'. The main content area displays the 'Winter Flowers' page, which includes a brief description of the winter season and a list of winter-blooming flowers. At the bottom of the main content area, there is a 'Comments' bar. This bar features a 'Comments' button, an 'Anonymous [anonymous]' link, a 'Log in' button, and a 'Add New Comment' button. A note at the bottom of the page states 'WebHelp output generated by <oxy>gen XML Author.'

Figure 288: The Layout of the Feedback-Enabled WebHelp System

After you log in, your name and user name are displayed in the **Comments** bar together with the **Log of** and **Edit** buttons.

Click the **Edit** button to open the **User Profile** dialog. In this dialog you can customize the following options:

- **Your Name** - you can use this field to edit the initial name that you used to create your user profile.
- **Your e-mail address** - you can use this field to edit the initial e-mail address that you used to create your profile.
- When to receive an e-mail:
 - When a comment is left on a page that you commented on.
 - When a comment is left on any topic in the Help system.
 - When a reply is left to one of my comments.
- **New Password** - allows you to enter a new password for your user account.



Note: The **Current Password** field from the top of the **User Profile** is mandatory in case you want to save the changes you make.

Search Tab

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on the following:

- The number of keywords found in a single page (the higher the number, the better).
- The context (for example, a word found in a title scores better than a word found in unformatted text). The search ranking order, sorted by relevance is as follows:
 - The search phrase is included in a meta keyword
 - The search phrase is in the title of the page
 - The search phrase is in bold text in a paragraph
 - The search phrase is in normal text in a paragraph

Growing Flowers

Content Search Index Flowers by Season / Summer Flowers

Keywords: grow flowers

Search

Results for: flowers, grow

[Lilac](#)
From Wikipedia, the free encyclopedia. Lilac (*Syringa*) flowers summer lilac is a genus of about 20–25 species of flowering plants in the olive family (Oleaceae), native to Europe and Asia. They... 

[Iris](#)
From Wikipedia, the free encyclopedia. Iris flowers spring iris is a genus of... Rubiaceae, native to the tropical and subtropical... 

Lilac

From Wikipedia, the free encyclopedia.

Lilac (*Syringa*) is a [genus](#) of about 20–25 species of [flowering](#) plants in the olive family ([Oleaceae](#)), native to Europe and Asia.



They are deciduous shrubs or small trees, ranging in size from 2–10 m tall, with stems up to 20–30 cm diameter. The leaves are opposite (occasionally in whorls of three) in arrangement, and their shape is simple and heart-shaped to broad lanceolate in most species, but pinnate in a few species (e.g. *S. protolaciniata*, *S. pinnatifolia*). The [flowers](#) are produced in spring, each flower being 5–10 mm in diameter with a four-lobed corolla, the corolla tube narrow, 5–20 mm long. The usual [flower](#) colour is a shade of purple (often a light purple or lilac), but white and pale pink are also found. The [flowers](#) grow in large [panicle](#), and in several species have a strong fragrance. Flowering varies between mid spring to early summer, depending on the species. The fruit is a dry, brown capsule, splitting in two at maturity to release the two winged seeds.

Related information
[Gardenia](#)

WebHelp output generated by  XML Author.

Figure 289: WebHelp Search with Stemming Enabled

Rules that are applied during a search include:

- The space character separates keywords (an expression such as *grow flowers* counts as two separate keywords: *grow* and *flowers*).
- Do not use quotes to perform an exact search for multiple word expressions (an expression such as "*grow flowers*", returns no results since it searches for two separate words).
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page (for example, content inside `keywords` elements weighs twice as much as content inside an `H1` HTML element).
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters count as a single word.
- Always search for words containing three or more characters (shorter words, such as *to* or *of* are ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

This output format is compatible with the following browsers:

- Internet Explorer (8 or newer)
- Chrome
- Firefox
- Safari
- Opera

Deployment of the WebHelp With Feedback System

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Editor requires the following system components:

- Apache Web Server running
 - MySQL server running
 - PHP Version 5.1.6 or later
 - PHP MySQL Support

Oxygen XML WebHelp system supports the following browsers: IE7+, Chrome 19+, Firefox 11+, Safari 5+, Opera 11+

Installation Instructions



Note: These instructions were written for XAMPP 1.7.7 with PHP 5.3.8 and for *phpMyAdmin* 3.4.5. Later versions of these packages may change the location or name of some options, however the following installation steps should remain valid and basically the same.

In case you have a web server configured with PHP, MySQL, you can deploy the WebHelp output directly. Otherwise, install XAMPP. XAMPP is a free and open source cross-platform web server solution stack package. It consists mainly of the Apache HTTP Server, MySQL database, and interpreters for scripts written in PHP.

Install XAMPP

1. Go to <https://www.apachefriends.org/download.html> and download XAMPP, for instance for a Windows system.
 2. Install it in C:\xampp.
 3. From the XAMPP control panel, start MySQL, and then Apache.
 4. Open <http://localhost/xampp/index.php> in your browser to check whether the HTTP server is working.

Create the WebHelp Feedback database

The WebHelp system needs a database to store user details and the actual feedback they provide. The following procedure creates a database for the feedback system and a MySQL user with privileges on that database. The feedback system uses these credentials to connect to the database.

Use *phpMyAdmin* to create a database:

1. Type `localhost` in your browser.
 2. In the left area, select: `phpMyAdmin`.
 3. Click `Databases` (in the right frame) and then create a `database`. You can give any name you want to your database, for example `comments`.
 4. Create a user with connection privileges for this database. In the **SQL** tab, paste the following text:

5. Change the *user_name* and the *user_password* values.
6. Under *localhost* in the right frame click *Privileges* and then at the bottom of the page click the **reload the privileges** link.

Deploying the WebHelp output

To deploy the WebHelp output, follow these steps:

1. Locate the directory of the HTML documents. Open `http://localhost/xampp/phpinfo.php` in your browser and see the value of the DOCUMENT_ROOT variable. In case you installed XAMPP in `C:\xampp`, the value of DOCUMENT_ROOT is `C:/xampp/htdocs`.
2. Copy the transformation output folder in the DOCUMENT_ROOT.
3. Rename it to a relevant name, for example, `webhelp_1`.
4. Open `http://localhost/webhelp_1/`. You are redirected to `http://localhost/webhelp_1/oxygen-webhelp/install/`.
 - Verify that the prerequisites are met.
 - Press **Start Installation**.
 - Configure the **Deployment Settings** section. Default values are provided, but you should adjust them as needed.
 - Configure the **MySQL Database Connection Settings** section. Use the details from the Create the WebHelp Feedback database section to fill-in the appropriate text boxes.

 **Warning:** Checking the **Create new database structure** option will overwrite any existing data in the selected database, if it already exists.

- If you are using a domain (such as *OpenLDAP* or *Active Directory*) to manage users in your organization, check the **Enable LDAP Authentication** option. This will allow you to configure the LDAP server, which will provide information and credentials for users who will access the WebHelp system. Also, this will allow you to choose which of the domain users will have administrator privileges.
- If the **Create new database structure** option is checked, the **Create WebHelp Administrator Account** section becomes available. Here you can set the administrator account data. The administrator is able to moderate new posts and manage WebHelp users.

The same database can be used to store comments for different WebHelp deployments. If a topic is available in more than one WebHelp deployments and there are comments associated with it, you can choose to display the comments in all deployments that share the database. To do this, enable the **Display comments from other products** option. In the **Display comments from** section a list with the deployments sharing the same database is displayed. Select the deployments allowed to share common feedback.

 **Note:** You can restrict the displayed comments of a product depending on its version. In case you have two products that use the same database and you restrict one of them to display comments starting from a certain version, the comments of the other product are also displayed from the specified version onwards.

- Press **Next Step**.
- Remove the installation folder from your web server.
- Click the link pointing to the index of the documentation, or visit: `http://localhost/webhelp_1/`.

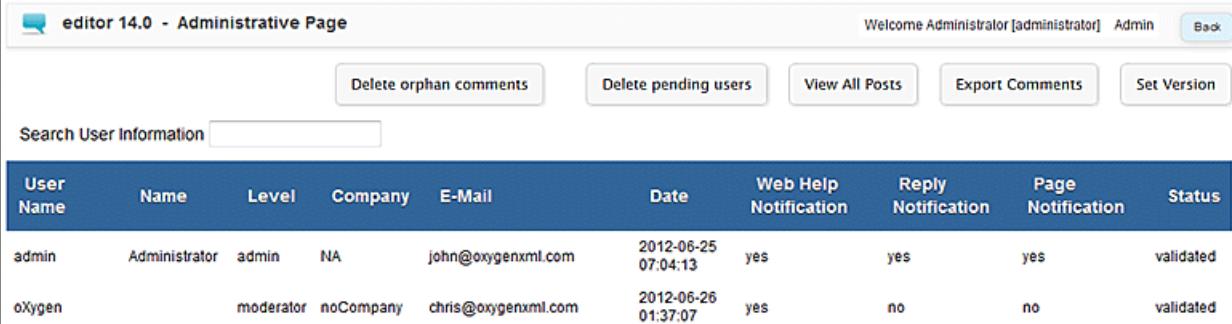
To test your system, create a user and post a comment. Check if the notification emails are delivered to your inbox.

 **Note:** To read debug messages generated by the system:

1. Enable *JScript* logging by doing one of the following:
 - Open the `log.js` file, locate the `var log = new Log(Level.NONE);` line, and change the logging level to: `Level.INFO`, `Level.DEBUG`, `Level.WARN`, or `Level.ERROR`.
 - Append `?log=true` to the WebHelp URL.
2. Inspect the PHP and Apache server log files.

Feedback System User Management

Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. To access the administrative page, select **Admin Panel** from the **Comments** bar.



The screenshot shows the 'editor 14.0 - Administrative Page' window. At the top right, it says 'Welcome Administrator [administrator] Admin Back'. Below that is a toolbar with buttons for 'Delete orphan comments', 'Delete pending users', 'View All Posts', 'Export Comments', and 'Set Version'. A search bar labeled 'Search User Information' is present. The main area is a table listing users:

User Name	Name	Level	Company	E-Mail	Date	Web Help Notification	Reply Notification	Page Notification	Status
admin	Administrator	admin	NA	john@oxygenxml.com	2012-06-25 07:04:13	yes	yes	yes	validated
oxygen		moderator	noCompany	chris@oxygenxml.com	2012-06-26 01:37:07	yes	no	no	validated

Figure 290: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp system. You can also view the details of each user and search through these details using the **Search User Information** filter.

The upper part of the page contains the following actions:

- **Delete Orphan Comments** - deletes comments associated with topics that are no longer available
- **Delete Pending Users** - deletes all unconfirmed users that registered more than a week ago
- **View All Posts** - allows you to view all posts associated with a product and version
- **Export Comments** - allows you to export in XML format all posts associated with a product and version
- **Set Version** - use this action to display comments starting from a particular version

To edit the details of a user, click the corresponding row. Use the **Edit User** dialog box to customize all the information associated with an user:

- **Name** - The user's full name
- **Level** - Use this field to modify the privilege level of the currently edited user. You can choose from:
 - **User** - regular user, able to post comments and receive e-mail notifications
 - **Moderator** - in addition to the regular **User** rights, this type of user has access to the **Admin Panel**. In the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system.
 - **Admin** - full administrative privileges. Can manage WebHelp-specific settings, users and their comments.
- **Company** - User's organization name
- **E-mail** - User's contact e-mail address. This is also the address where the WebHelp system sends notifications:
 - **WebHelp Notification** - when enabled, the user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system
 - **Reply Notification** - when enabled, the user receives notifications when comments are posted as a reply to one of his or hers comments
 - **Page Notification** - when enabled, the user receives notifications when comments are posted on a topic where he or she posted a comment
- **Date** - User registration date
- **Status** - Specifies the status of the currently edited user:
 - **Created** - the user is created but does not have any rights over the feedback-enabled WebHelp system
 - **Validated** - the user is able to use the feedback-enabled WebHelp system
 - **Suspended** - the user has no rights over the feedback-enabled WebHelp system

WebHelp Mobile System Description

To further improve its ability to create online documentation, Oxygen XML Editor offers support to transform DocBook And DITA documents into *Mobile WebHelp* systems. This feature generates an output that works on multiple platforms (Android, iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices. All the specific touch screen gestures are supported. The functionality of the desktop WebHelp layout is preserved, is organized in an intuitive layout, and offers table of contents, search capabilities, and index navigation.



Figure 291: Mobile WebHelp

Context-Sensitive WebHelp System

Context-sensitive help systems assist users by providing specific informational topics for certain components of a user interface, such as a button or window. This mechanism works based on mappings between a unique ID defined in the topic and a corresponding HTML page.

When WebHelp output is generated by Oxygen XML Editor, the transformation process produces an XML mapping file called `context-help-map.xml` and copies it in the output folder. This XML file maps an ID to a corresponding HTML page like:

```
<map productID="oxy-webhelp" productVersion="1.1">
  <appContext helpID="annotations-view" path="topics/annotations-view.html"/>
  <appContext helpID="button-editor" path="concepts/button-editor.html"/>
</map>
```

where:

- `helpID` - unique ID provided by a topic from two possible sources:
 - the `resourceid` element set to it in the `prolog` section:

```
<prolog>
  <resourceid id="context-sensitive-help-system"/>
</prolog>
```



Note: If you need different parts of the application (for instance, dialog boxes, views, or editing modes) to open the same contextual help topic, all of the context ID values should be included in the same DITA topic file. For example, if you need both a dialog box and a view to open the same WebHelp page, you can assign different resource ID in the same DTIA topic.

```
<prolog>
  <resourceid id="dialog1"/>
```

```
<resourceid id="view1"/>
</prolog>
```

- the `id` attribute set on the topic root element



Important: You should ensure that these defined IDs are unique in the context of the entire DITA project. If the IDs are not unique, the transformation scenario will display warning messages in the transformation console output. In this case the help system will not work properly.

- `path` - path to a corresponding WebHelp page. This path is relative to the location of the `context-help-map.xml` mapping file.
- `productID` - ID of the product for which you are writing documentation. Applicable only if you are using WebHelp with Feedback transformations.
- `productVersion` - version of the product for which you are writing documentation. Applicable only if you are using WebHelp with Feedback transformations.

There are two ways of implementing context-sensitive help in your application:

- The XML mapping file can be loaded by a PHP script on the server side. The script receives the context ID value and will look it up in the XML file.
- Invoke one of the WebHelp system files `index.html` or `index_frames.html` and pass them the `contextId` parameter with a specific value. The WebHelp system will automatically open the help page associated with the value of the `contextId` parameter.

The following example will open a *frameless* version of the WebHelp system showing the page associated with the id `dialog1ID`:

```
index.html?contextId=dialog1ID
```

The following example will open a *frameset* version of the WebHelp system showing the page associated with the id `view1ID`:

```
index_frames.html?contextId=view1ID
```

Customizing the WebHelp Systems

This section contains various customizations that you can make to the output of your WebHelp transformation.

To change the overall appearance of the WebHelp output, you can use the visual [WebHelp Skin Builder tool](#), which does not require knowledge of CSS language.

If you are familiar with CSS and coding, this section includes topics that explain how you can customize your WebHelp system, such as how to improve the appearance of the Table of Contents, add logo images in the title area, remove the navigation buttons, and add custom headers and footers.

The WebHelp Skin Builder

The **WebHelp Skin Builder** is a simple, easy-to-use tool, specially designed to assist users to visually customize the look and feel of the WebHelp output. It is implemented as an online tool hosted on the Oxygen XML Editor website and allows you to experiment with different styles and colors over an inert documentation sample.

To be able to use the **Skin Builder**, you need:

- An Internet connection and unrestricted access to Oxygen XML Editor website.
- A later version web browser.

To start the **Skin Builder**, do one of the following:

- From a web browser navigate to <http://www.oxygenxml.com/webhelp-skin-builder>.
- From the Oxygen XML Editor in the **Skins tab**, click the **Online preview** link. In the upper section of the preview, click the **Select Skin** button, then choose **Customize Skin**.

The Skin Builder Layout

The left side panel of the *Skin Builder* is divided into 3 sections:

- **Actions** - contains two buttons:
 - **Import** - allows you to load a CSS stylesheet and applies it over the documentation sample.
 - **Export** - saves all properties as a CSS file.
- **Settings** - contains the **Highlight selection** checkbox which helps you identify the areas affected by a particular element customization:
 - When hovering an item in the customizable elements menu, the affected sample area is highlighted with a dotted blue border.
 - When an item in the customizable elements menu is selected, the affected sample area is highlighted with a solid red border.
- **Customize** - provides a series of customizable elements organized under four main categories:
 - Header
 - TOC Area
 - Vertical Splitter
 - Content

For each customizable element you can alter properties like background color or font face. Any alteration made in the customizable elements menu is applied in real time over the sample area.

Create a Customization Skin

- The starting point can be either one of the predefined skins or a CSS stylesheet applied over the sample using the **Import** button.
- Use the elements in the **Customize** section to set properties that modify the skin's look. By default, all customizable elements display a single property, but you can make more properties visible if you click the **+ Add** button and choose from the available ones.



Note: If you want to revert a setting of a particular property to its initial value, press the **Reset** button.

- When you are happy with the skin customization you have made, press the **Export** button. All settings will be saved in a CSS file.

Apply a Customization Skin to a DITA Map to WebHelp Transformation Scenario

- Start Oxygen XML Editor.
- Load the DITA Map you want to produce as a WebHelp output.
- Edit a *DITA Map to WebHelp*-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins tab**.
- Execute the transformation to obtain the WebHelp output.

Apply a Customization Skin to a DocBook to WebHelp Transformation Scenario

- Start Oxygen XML Editor.
- Load the DocBook file you want to produce as a WebHelp output.

- Edit a *DocBook to WebHelp*-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins** tab.
- In the **Parameters** tab, set the `webhelp.skin.css` parameter to point to the previously exported CSS.
- To customize the logo, use the following parameters:
 - `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
 - `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

Automating the WebHelp Output

Oxygen XML WebHelp plugin allows you to use a command line interface script to obtain WebHelp output from DITA and DocBook documents. Note that the Oxygen XML WebHelp plugin is a standalone product with its own licensing terms and cannot be used with a Oxygen XML Editor license.

The WebHelp output files created with the Oxygen XML WebHelp plugin are the same as the output files produced when you run DITA or DocBook to WebHelp transformation scenarios from within Oxygen XML Editor.

When an automated process is required due to the amount of output needed, do the following:

1. Install the Oxygen XML WebHelp plugin
2. Acquire a Oxygen XML WebHelp license from http://www.oxygenxml.com/buy_webhelp.html

Oxygen XML WebHelp Plugin for DITA

To transform DITA documents using the Oxygen XML WebHelp plugin, first integrate the plugin with the DITA Open Toolkit. The purpose of the integration is to add to the DITA Open Toolkit the following transformation types:

- **webhelp** - the transformation that produces *WebHelp* output for desktop
- **webhelp-feedback** - the transformation that produces feedback-enabled *WebHelp* for desktop
- **webhelp-mobile** - the transformations that produces *WebHelp* output for mobile devices

Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit

The requirements of the Oxygen XML WebHelp plugin for the DITA Open Toolkit are:

- Java Virtual Machine 1.6 or later
- DITA Open toolkit 1.6.x, 1.7.x, 1.8, or 2.0 (Full Easy Install)
- Saxon 9.1.0.8

To integrate the Oxygen XML WebHelp plugin with the DITA Open Toolkit, follow these steps:

1. Download and install a *Java Virtual Machine* 1.6 or later.
2. Download and install *DITA Open Toolkit* 1.6.x, 1.7.x, 1.8, or 2.0.
3. Navigate to the `plugins` directory located in the installation directory of the DITA Open Toolkit.
4. Copy the `com.oxygenxml.webhelp` and `com.oxygenxml.highlight` directories inside the `plugins` directory. The `com.oxygenxml.highlight` directory add syntax highlight capabilities to your WebHelp output for codeblock sections that contain source code snippets (XML, Java, JavaScript etc.).
5. If you are using DITA-OT version 2.0, the WebHelp plugin contains a `plugin_2.x.xml` which needs to be renamed to `plugin.xml`.
6. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.
7. Go to <http://sourceforge.net/projects/saxon/files/Saxon-B/9.1.0.8/>, download and unzip the `processor.saxonb9-1-0-8j.zip` file (contains the Saxon 9.1.0.8).

Registering the Oxygen XML WebHelp Plugin

To register the Oxygen XML WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Open the `[DITA-OT-install-dir]/plugins/com.oxygenxml.webhelp` directory and create a file called `licensekey.txt`.
2. In this file, copy your license key which you purchased for your Oxygen XML WebHelp plugin.

The WebHelp transformation process reads the Oxygen XML Editor license key from this file. In case the file does not exist, or it contains an invalid license, an error message will be displayed.

Running a DITA Transformation Using the Oxygen XML WebHelp Plugin

To run a DITA to WebHelp (**webhelp**, **webhelp-feedback**, **webhelp-mobile**) transformation using the Oxygen XML WebHelp plugin, use:

- The `dita.bat` script file for Windows based systems.
- The `dita.sh` script file for Unix/Linux based systems.



Note: You can call these files in an automated process or from the command line.

The `dita.bat` and the `dita.sh` files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate an WebHelp system, customize them to match the paths to the JVM, DITA Open Toolkit and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- `JVM_INSTALL_DIR` - specifies the path to the Java Virtual Machine installation directory on your disk.
- `DITA_OT_INSTALL_DIR` - specifies the path to DITA Open Toolkit installation directory on your disk.
- `SAXON_9_DIR` - specifies the path to the directory on your disk where you unzipped the Saxon 9 archive files.
- `TRANSTYPE` - specifies the type of the transformation you want to execute. You can set it to `webhelp`, `webhelp-feedback` and `webhelp-mobile`.
- `DITA_MAP_BASE_DIR` - specifies the path to the directory where the input DITA Map file is located.
- `DITAMAP_FILE` - specifies the input DITA Map file.
- `DITAVAL_FILE` - specifies the `.ditaval` input filter that the transformation process applies to the input DITA Map file.
- `DITAVAL_DIR` - specifies the path to the directory where the `.ditaval` file is located.
- `Doutput.dir` - specifies the output directory of the transformation.

The `-Dargs.filter` and the `-Ddita.input.valfile` parameters are optional.

Additional Oxygen XML WebHelp Plugin Parameters for DITA

You are able to append the following parameters to the command line that runs the transformation:

- `-Dwebhelp.copyright` - the copyright note that is added in the footer of the Table of Contents frame;
- `-Dwebhelp.footer.file` - specifies the location of a well-formed XHTML file containing your custom footer for the document body. Corresponds to the `WEBHELP_FOOTER_FILE` XSLT parameter. The fragment must be a well-formed XHTML, with a single root element. As a common practice, place all the content into a `<div>` element;
- `-Dwebhelp.footer.include` - specifies whether the content of file set in the `-Dwebhelp.footer.file` is used as footer in the WebHelp pages. Its values can be `yes`, or `no`;
- `-Dwebhelp.product.id` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It represents a short name of the documentation target (product). All the user comments that are posted in the WebHelp output pages and are added in the comments database are bound to this product ID;



Note: You can deploy documentation for multiple products on the same server.

- `-Dwebhelp.product.version` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.



Note: Multiple documentation versions can be deployed on the same server.

In case you need to further customize the transformation process, you are able to append other DITA-OT parameters as well. Any parameter that you want to append must follow the `-D` model of the above parameters. For example, to append the `args.hdr` parameter, use:

```
-Dargs.hdr=[HEADER_FILE_DIR]
```

where [HEADER_FILE_DIR] is the location of the directory that contains the header file.

Database Configuration for DITA WebHelp with Feedback

If you run the **webhelp-feedback** transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at [DITA_MAP_BASE_DIR] /out / [TRANSFORM_TYPE] /oxygen-webhelp/resources. The `installation.html` file is created by the transformation process.

Oxygen XML WebHelp Plugin for DocBook

To transform DocBook documents using the Oxygen XML WebHelp plugin, first integrate the plugin with the DocBook XSL distribution. The purpose of the integration is to add to the DocBook XSL distribution the following transformation types:

- **webhelp** - the transformation that produces *WebHelp* output for desktop
- **webhelp-feedback** - the transformation that produces feedback-enabled *WebHelp* for desktop
- **webhelp-mobile** - the transformations that produces *WebHelp* output for mobile devices

Integrating the Oxygen XML WebHelp Plugin with the DocBook XSL Distribution

The WebHelp plugin transformations run as an ANT build script. The requirements are:

- ANT 1.8 or later
- Java Virtual Machine 1.6 later
- DocBook XSL 1.78.1 later
- Saxon 6.5.5
- Saxon 9.1.0.8

To integrate the Oxygen XML WebHelp plugin with the DocBook XSL distribution, follow these steps:

1. Download and install a [Java Virtual Machine](#) 1.6 or later.
2. Download and install [ANT 8.0](#) or later.
3. Download and unzip on your computer the DocBook XSL distribution.
4. Unzip the Oxygen XML WebHelp distribution package in the DocBook XSL installation directory.
The DocBook XSL directory now contains a new subdirectory named `com.oxygenxml.webhelp` and two new files, `oxygen_custom.xsl` and `oxygen_custom_html.xsl`.
5. Download and unzip [saxon6-5-5.zip](#) on your computer.
6. Download and unzip [saxonb9-1-0-8j.zip](#) on your computer.

Registering the Oxygen XML WebHelp Plugin

To register the Oxygen XML WebHelp plugin for the DocBook XSL distribution, follow these steps:

1. Create a `.txt` file named `license` in the `com.oxygenxml.webhelp` subdirectory of the DocBook XSL directory.
2. In this file, copy the license key, which you purchased for your Oxygen XML WebHelp plugin.
The WebHelp transformation process reads the Oxygen XML Editor license key from this file. If the file does not exist, or it contains an invalid license, an error message is displayed.

Running a DocBook Transformation Using the WebHelp Plugin

To run a DocBook to WebHelp (**webhelp**, **webhelp-feedback**, **webhelp-mobile**) transformation using the Oxygen XML WebHelp plugin, use:

- The `docbook.bat` script file for Windows based systems.
- The `docbook.sh` script file for Unix/Linux based systems.



Note: You can call these files in an automated process or from the command line.

The docbook.bat and the docbook.sh files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate an WebHelp system, customize them to match the paths to the JVM, DocBook XSL distribution and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- `JVM_INSTALL_DIR` - specifies the path to the Java Virtual Machine installation directory on your disk.
- `ANT_INSTALL_DIR` - specifies the path to the installation directory of ANT.
- `SAXON_6_DIR` - specifies the path to the installation directory of Saxon 6.5.5.
- `SAXON_9_DIR` - specifies the path to the installation directory of Saxon 9.1.0.8.
- `DOCBOOK_XSL_DIR` - specifies the path to the installation directory of the DocBook XSL distribution.
- `TRANSTYPE` - specifies the type of the transformation you want to execute. You can set it to `webhelp`, `webhelp-feedback` and `webhelp-mobile`.
- `INPUT_DIR` - specifies the path to the input directory, containing the input XML file.
- `XML_INPUT_FILE` - specifies the name of the input XML file.
- `OUTPUT_DIR` - specifies the path to the output directory where the transformation output is generated.
- `DOCBOOK_XSL_DIR_URL` - specifies the path to the directory of the DocBook XSL distribution in URL format.

Additional Oxygen XML WebHelp Plugin Parameters for DocBook

You are able to append the following parameters to the command line that runs the transformation:

- `-Dwebhelp.copyright` - the copyright note (a text string value) that is added in the footer of the table of contents frame (the left side frame of the WebHelp output);
- `-Dwebhelp.footer.file` - specifies the location of a well-formed XHTML file containing your custom footer for the document body. Corresponds to the `WEBHELP_FOOTER_FILE` XSLT parameter . The fragment must be an well-formed XHTML, with a single root element. As a common practice, place all the content inside a `<div>` element;
- `-Dwebhelp.footer.include` - specifies whether the content of file set in the `-Dwebhelp.footer.file` is used as footer in the WebHelp pages. Its values can be `yes`, or `no`;
- `-Dwebhelp.product.id` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It represents a short name of the documentation target (product). All the user comments that are posted in the WebHelp output pages and are added in the comments database are bound to this product ID;



Note: You can deploy documentation for multiple products on the same server.

- `-Dwebhelp.product.version` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.



Note: Multiple documentation versions can be deployed on the same server.

In case you need to further customize your transformation, other DocBook XSL parameters can be appended. Any parameter that you want to append must follow the `-D` model of the above parameters. For example, you can append the `html.stylesheet` parameter in the following form:

```
-Dhtml.stylesheet=/path/to/directory/of/stylesheet.css
```

Database Configuration for DocBook WebHelp with Feedback

In case you run the **webhelp-feedback** transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at `[OUTPUT_DIR]/oxygen-webhelp/resources/installation.html`. The `installation.html` file is created by the transformation process.

Localizing the Email Notifications of the WebHelp with Feedback System

The WebHelp with Feedback system uses emails to notify users when comments are posted. These emails are based on templates stored in the WebHelp directory. The default messages are in English, French, German, and Japanese. These

messages are copied into the WebHelp system deployment directory during the execution of the corresponding transformation scenario.

We'll suppose that you want to localize the emails into Dutch. Follow these steps:

DocBook to WebHelp with Feedback

- create the following directory:
[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl
 - copy all English template files from
[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en and paste them into the directory you just created
 - edit the HTML files from the
[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl directory and translate the content into Dutch
 - start Oxygen XML Editor and edit the *WebHelp with Feedback* transformation scenario
 - in the **Parameters** tab look for the `110n.gentext.default.language` parameter and set its value to the appropriate language code. In our example, use the value nl for Dutch
-  **Note:** If you set the parameter to a value such as `LanguageCode-CountryCode` (for example, `en-us`), the transformation scenario will only use the language code
- execute the transformation scenario to obtain the *WebHelp with Feedback* output

DITA to WebHelp with Feedback

- create the following directory:
[OXYGEN_DIR]\frameworks\dita\DTIA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl
 - copy all English template files from
[OXYGEN_DIR]\frameworks\dita\DTIA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en and paste them into the directory you just created
 - edit the HTML files from the
[OXYGEN_DIR]\frameworks\dita\DTIA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl directory and translate the content into Dutch
 - start Oxygen XML Editor and edit the *WebHelp with Feedback* transformation scenario
 - in the **Parameters** tab look for the `args.default.language` parameter and set its value to the appropriate language code. In our example, use the value nl for Dutch
-  **Note:** If you set the parameter to a value such as `LanguageCode-CountryCode` (for example, `en-us`), the transformation scenario will only use the language code
- execute the transformation scenario to obtain the *WebHelp with Feedback* output

Adding Videos in the Output

Videos can be included and played in all HTML5-based output formats (like *WebHelp*). For example, to add a YouTube video in the WebHelp output generated from DITA or DocBook documents, follow the procedures below.

Adding Videos to WebHelp Generated from DITA Maps

- Edit the DITA topic to reference the video using an `object` element like in the following example:
- ```
<object outputclass="video">
 <param name="src" value="http://www.youtube.com/watch/v/VideoName" />
</object>
```
- Apply a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output

## Adding Videos to WebHelp Generated from DocBook

- Edit the DocBook document and reference the video using an `mediaobject` element like in the following example:

```
<mediaobject>
 <videoobject>
 <videodata fileref="http://www.youtube.com/watch/v/VideoName" />
 </videoobject>
</mediaobject>
```

- Apply a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output

## CSS Customizations

Adding your own CSS stylesheet enables you to customize the WebHelp output. To do this, edit the transformation scenario and set the `args.css` parameter to point to your custom CSS document. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.

### Table of Contents Customization

The appearance of the selected item in the Table of Contents can be enhanced. To highlight the background of the selected item, go to the output folder of the WebHelp transformation and locate the `toc.css` files in the **oxygen-webhelp > resources > skins > desktop** and **oxygen-webhelp > resources > skins > desktop-frames** folders. Open them, find the `menuItemSelected` class, and change the value of the `background` property.

 **Note:** Also, you can overwrite the same value from your own CSS.

### Changing the Icons in a WebHelp Table of Contents

You can change the icons that appear in a WebHelp table of contents by assigning new image files in a custom CSS file. By default, the icons for the WebHelp table of contents are defined with the following CSS codes (the first example is the icon that appears for a collapsed menu and the second for an expanded menu):

```
.hasSubMenuClosed{
 background: url('../img/book_closed16.png') no-repeat;
 padding-left: 16px;
 cursor: pointer;
}

.hasSubMenuOpened{
 background: url('../img/book_opened16.png') no-repeat;
 padding-left: 16px;
 cursor: pointer;
}
```

To assign different icons use the following procedure:

1. Create a custom CSS file that assigns your desired icons to the `.hasSubMenuClosed` and `.hasSubMenuOpened` properties.

```
.hasSubMenuClosed{
 background: url('TOC-my-closed-button.png') no-repeat;
}

.hasSubMenuOpened{
 background: url('TOC-my-opened-button.png') no-repeat;
}
```

2. It is recommended that you store the image files in the same directory as the default icons:  
`[OXYGEN_INSTALL_DIR]\frameworks\dita\DIITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img\`
3. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.
  - a) For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes`.
  - b) For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.

## Adding a Logo Image in the Title Area

You are able to customize the title of your WebHelp output by using a custom CSS.

For example, to add a logo image before the title, use the following code:

```
h1:before {
 display:inline;
 content:url('../img/myLogoImage.gif');
}
```

In the example above, **myLogoImage.gif** is an image file that you place in the `[OXYGEN_DIR]\frameworks\dita\OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img` directory, thus it is copied automatically by the WebHelp transformation to the output directory.

## Removing the Previous/Next Links from Each WebHelp Page

The **Previous** and **Next** links that are created at the top area of each WebHelp page can be hidden with the following CSS code:

```
.navparent, .navprev, .navnext {
 visibility:hidden;
}
```

 **Tip:** Add the above code in a custom CSS stylesheet and in a WebHelp transformation scenario, set the **args.css** parameter to reference the path of this CSS stylesheet.

## Adding Custom Headers and Footers

In the transformation scenario, you can use the `args.hdr` and `args.ftr` parameters to point to resources that contain your custom HTML `<div>` blocks. These are included in the header and footer of each generated topic.

To hide the horizontal separator line between the content and footer, edit the DITA transformation scenario and configure the following parameters:

- The `args.css` parameter to reference a CSS file containing the following CSS snippet:

```
.footer_separator {
 display:none;
}
```

- The `args.copycss` parameter set to `true`.

## Change numbering styles for ordered lists

Ordered lists (`ol`) are usually numbered in XHTML output using numerals. If you want to change the numbering to alphabetical, do the following:

1. Define a custom `outputclass` value and set it as an attribute of the ordered list, as in the following example:

```
<ol outputclass="number-alpha">
 A
 B
 C

```

2. Add the following code snippet in a custom CSS file:

```
ol.number-alpha{
 list-style-type:lower-alpha;
}
```

3. Edit the DITA transformation scenario and configure the following parameters:

- `args.css` parameter to reference the custom CSS file appended earlier
- `args.copycss` parameter set to `true`.

## WebHelp Runtime Additional Parameters

A deployed WebHelp system can accept the following GET parameters:

- `log` - The value can be `true` or `false` (default value). When set to `true`, it enables JavaScript debugging.
- `contextId` - The WebHelp JavaScript engine will look up the value of this parameter in the mapping file and load the corresponding HTML help page.
- `toc.visible` - The value can be `true` (default value) or `false`. When to `false`, the table of contents will be collapsed when you load the WebHelp page.

---

# Chapter

# 12

---

## Querying Documents

---

**Topics:**

- *Running XPath Expressions*
- *Working with XQuery*

This chapter shows how to query XML documents in Oxygen XML Editor with XPath expressions and the XQuery language.

## Running XPath Expressions

This section covers the views, toolbars, and dialogs in Oxygen XML Editor, dedicated to running XPath expressions.

### What is XPath

XPath is a language for addressing specific parts of an XML document. XPath, like the Document Object Model (DOM), models an XML document as a tree of nodes. An XPath expression is a mechanism for navigating through and selecting nodes from the XML document. An XPath expression is, in a way, analogous to an SQL query used to select records from a database.

There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node.

XPath defines a library of standard functions for working with strings, numbers and boolean expressions.

- `child::*` - Selects all children of the root node.
- `./name` - Selects all elements having the name "name", descendants of the current node.
- `/catalog/cd[price>10.80]` - Selects all the `cd` elements that have a `price` element with a value larger than 10.80.

To find out more about XPath, go to <http://www.w3.org/TR/xpath>.

### Oxygen XPath Toolbar

XPath is a query language for selecting nodes from an XML document. To use XPath expressions effectively, you need a good understanding of [the XPath Core Function Library](#).

#### The XPath Toolbar

Oxygen XML Editor provides an XPath toolbar to let you query XML documents fast and easy using XPath expressions.



**Figure 292: The XPath Toolbar**

You can choose the XPath version from the drop-down menu available in the left side of the toolbar. Available options include XPath 1.0, XPath 2.0, XPath 2.0 SA, XPath 3.0, XPath 3.0 SA.

- Note:** XPath 2.0 SA and XPath 3.0 SA use the Saxon EE [XML Schema version](#) option.
- Note:** The results returned by XPath 2.0 SA and XPath 3.0 SA have a location limited to the line number of the start element (there are no column information and no end specified).
- Note:** Oxygen XML Editor uses Saxon to execute XPath 3.0 expressions, but implements a part of the 3.0 functions. When using a function that is not implemented, Oxygen XML Editor can return a compilation error.

**XPath scope** menu - Oxygen XML Editor allows you to define a scope on which the XPath operation will be executed. You can choose where the XPath expression will be executed:

- **Current file** - current selected file only
- **Project** - all the files in the project
- **Selected project resources** - the files selected in the project.
- **All opened files** - all files opened in the application

- **Current DITA Map hierarchy** - all resources referenced in the currently selected DITA map, opened in the DITA Maps Manager view
- **Opened archive** - files open in the *Archive Browser* view
- **Working sets** - the selected working sets

At the bottom of the scope menu there are available the following scope configuration actions:

- **Configure XPath working sets** - allows you to configure and manage collections of files and folders, encapsulated in logical containers called *working sets*
- **XPath file filter** - you can filter the files from the selected scope on which the XPath expression will be executed. By default the XPath expression will be executed only on XML files, but you can also define a set of patterns that will filter out files from the current scope. If you select the **Include archive** option, the XPath expression will be also executed on the files in any archive (including EPUB and DocX) found at the current scope.

The following actions are available in the **Settings** menu:

- **Update on caret move** - when enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed
- **Evaluate as you type** - when you select this option, the XPath expression you are composing is evaluated in real time
 

**Note:** The **Evaluate as you type** option and the automatic validation are disabled when you edit *huge documents* or when the scope is other than **Current file**
- **Options** - opens the Preferences page of the currently selected processing engine
- Note:** During the execution of an XPath expression, the XPath toolbar displays a stop button . Press this button to stop the XPath execution.

When you type expressions longer than 60 characters, a dialog pops up and offers you the possibility to switch to the **XPath builder** view.

## The XPath/XQuery Builder View

The **XPath/XQuery Builder** view allows you to compose complex XPath and XQuery expressions and execute them over the currently edited XML document. For XPath 2.0 / 3.0, or XQuery expressions, you are able to use the `doc()` function to specify the source file over which the expressions are executed. When you connect to a database, the expressions are executed over that database. If you are using the **XPath/XQuery Builder** view and the current file is an XSLT document, Oxygen XML Editor executes the expressions over the XML document in the associated scenario.

To open the **XPath/XQuery Builder** view, go to **Window > Show View > XPath/XQuery Builder**.

The upper part of the view contains the following actions:

- a drop-down list that allows you to select the type of the expression you want to execute. You can choose between:
  - XPath 1.0 (Xerces-driven)
  - XPath 2.0, XPath 2.0SA, XPath 3.0, XPath 3.0SA, XQuery 1.0, XQuery 3.0, Saxon-HE XQuery, Saxon-PE XQuery, or Saxon-EE XQuery (all of them are Saxon-driven)
  - custom connection to XML databases that can execute XQuery expressions

**Note:** The results returned by XPath 2.0 SA and XPath 3.0 SA have a location limited to the line number of the start element (there are no column information and no end specified).

**Note:** Oxygen XML Editor uses Saxon to execute XPath 3.0 expressions. Because Saxon implements a part of the 3.0 functions, when using a function that is not implemented, Oxygen XML Editor returns a compilation error.

-  **Execute XPath** button - press this button to start the execution of the XPath or XQuery expression you are editing. The result of the execution is displayed in the **Results view** in a separate tab
- **Favorites** button - allows you to save certain expressions that you can later reuse. To add an expression as favorite, press the star button and enter a name under which the expression is saved. The star turns yellow to confirm that the expression was saved. Expand the drop-down list next to the star button to see all your favorites. Oxygen XML Editor automatically groups favorites in folders named after the method of execution
-  **History** drop-down box - keeps a list of the last 15 executed XPath or XQuery expressions. Use the  **Clear history** action from the bottom of the list to remove them
-  **Settings** drop-down menu - contains three options:
  -  **Update on caret move** - when enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed
  -  **Evaluate as you type** - when you select this option, the XPath expression you are composing is evaluated in real time

 **Note:** The  **Evaluate as you type** option and the automatic validation are disabled when you edit *huge documents* or when the scope is other than **Current file**
-  **Options** - opens the Preferences page of the currently selected processing engine
- **XPath scope** menu - Oxygen XML Editor allows you to define a scope on which the XPath operation will be executed. You can choose where the XPath expression will be executed:
  -  **Current file** - current selected file only
  -  **Project** - all the files in the project
  -  **Selected project resources** - the files selected in the project.
  -  **All opened files** - all files opened in the application
  -  **Current DITA Map hierarchy** - all resources referenced in the currently selected DITA map, opened in the DITA Maps Manager view
  -  **Opened archive** - files open in the *Archive Browser* view
  -  **Working sets** - the selected working sets

At the bottom of the scope menu there are available the following scope configuration actions:

-  **Configure XPath working sets** - allows you to configure and manage collections of files and folders, encapsulated in logical containers called *working sets*
-  **XPath file filter** - you can filter the files from the selected scope on which the XPath expression will be executed. By default the XPath expression will be executed only on XML files, but you can also define a set of patterns that will filter out files from the current scope. If you select the **Include archive** option, the XPath expression will be also executed on the files in any archive (including EPUB and DocX) found at the current scope.

```

XPath/XQuery Builder - authors.xquery
XQuery 1.0 ▶
Scope: Current File ▾
1 (: You can activate the content completion by pressing the Ctrl+Space keys. :)
2 xquery version "1.0";
3
4 (: Namespace for the <oXygen/> custom functions and variables :)
5 declare namespace oxy="http://www.oxygenxml.com/xquery/functions";
6
7 (: The URI of the document that is to be queried :)
8 declare variable $oxy:document-to-query as xs:string := "books.xml";
9
10 (: Queries an XML document for authors
11 declare function oxy:list-authors($d
12
13 let $library := doc($document)
14 let $seq := $library//author
15 let $distinct := distinct-values($seq)
16
17 for $a in $distinct
18 return
19 <author>
20 <name> {$a} </name>
21 {
22 for $book in $library/library/publications
23 where (compare($book/author, $a)
24 return $book/title
25 }
26 </author>
27 };
28
29 <author_list>
30 {oxy:list-authors($oxy:document-to-query)}
31 </author_list>
32

```

**Figure 293: The XPath/XQuery Builder View**

While you edit an XPath or XQuery expression, Oxygen XML Editor assists you with the following features:

- Content Completion Assistant - It offers context-dependent proposals and takes into account the cursor position in the document you are editing. The set of functions proposed by the **Content Completion Assistant** also depends on the engine version. Select the engine version from the drop-down menu available in the toolbar.
  - Syntax highlight - allows you to identify the components of an expression. To customize the colors of the components of the expression, [open the Preferences dialog box](#) and go to **Editor > Colors**.
  - Automatic validation of the expression as you type.
- Note:** When you type invalid syntax a red serrated line underlines the invalid fragments.
- Function signature and documentation balloon, when the cursor is located inside a function.

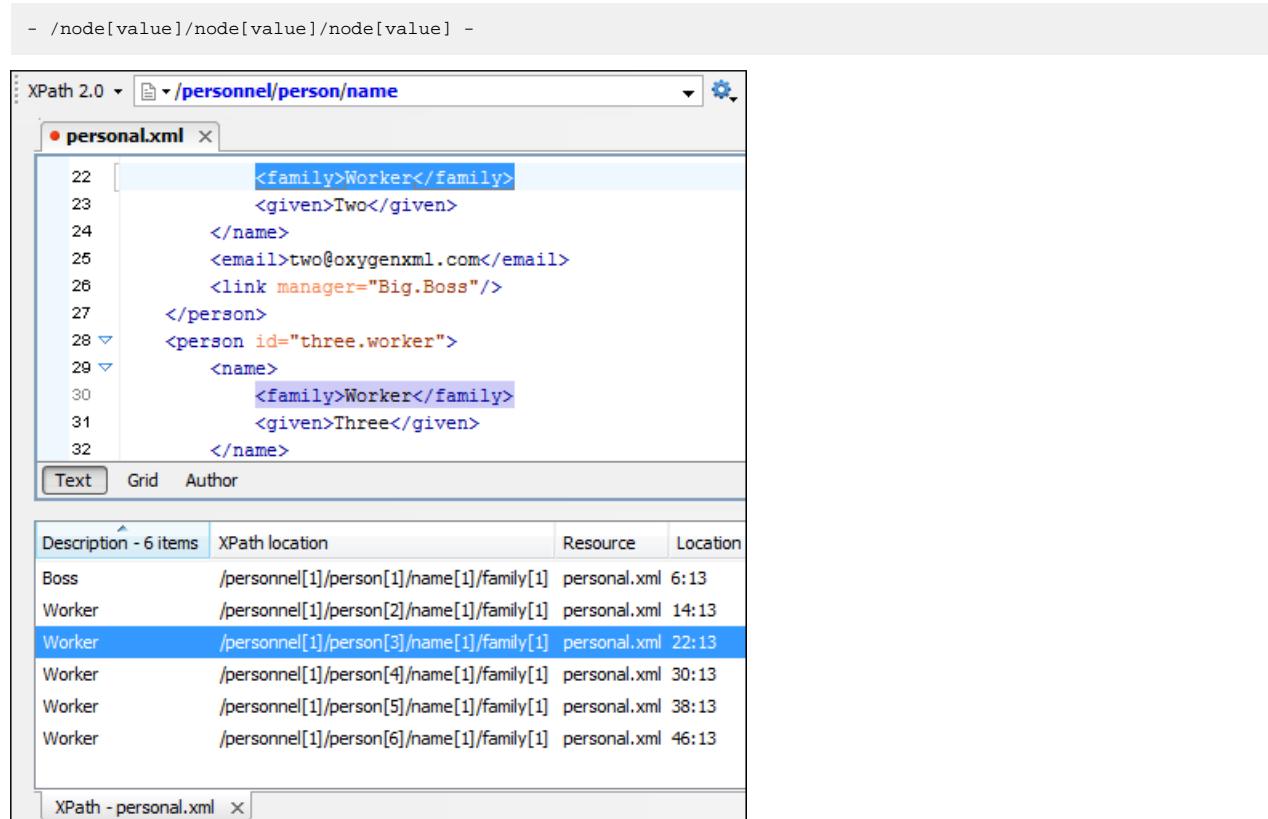
The [usual edit actions](#) **Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo** are available in the pop-up menu of the top editable part of the view.

## XPath Results View

When you run an XPath expression, Oxygen XML Editor displays the results of its execution in the **Results View**. This view contains five columns:

- **Description** - Holds the result that Oxygen XML Editor displays when you run an XPath expression.
- **XPath location** - Holds the path to the matched node.
- **Resource** - Holds the name of the document on which you run the XPath expression.
- **System ID** - Holds the path to the document itself.
- **Location** - Holds the location of the result in the document.

To arrange the results depending on a column click on its header. To group the results by their resource, or by their system id, right click the header of any column in the results view and select **Group by "Resource"** or **Group by "System ID"**. If no information regarding location is available, Oxygen XML Editor displays **Not available** in the **Location** column. Oxygen XML Editor displays the results in a valid XPath expression format.



**Figure 294: XPath results highlighted in editor panel with character precision**

The following snippets are taken from a DocBook book based on the DocBook XML DTD. The book contains a number of chapters. To return all the chapter nodes of the book, enter `//chapter` in the XPath expression field and press **(Enter)**. This action returns all the chapter nodes of the DocBook book in the **Results View**. Click a record in the **Results View** to locate and highlight its corresponding chapter element and all its children nodes in the document you are editing.

To find all `example` nodes contained in the `sect2` nodes of a DocBook XML document, use the following XPath expression: `//chapter/sect1/sect2/example`. Oxygen XML Editor adds a result in the **Results View** for each `example` node found in any `sect2` node.

For example, if the result of the above XPath expression is:

```
- /chapter[1]/sect1[3]/sect2[7]/example[1]
```

it means that in the edited file the `example` node is located in the first chapter, third section level one, seventh section level 2.

## Catalogs

The evaluation of the XPath expression tries to resolve the locations of documents referenced in the expression through the [XML catalogs](#). These catalogs are [configured in the Preferences](#) pages and [the current XInclude preferences](#).

Let's take as an example the evaluation of the `collection(URIofCollection)` function (XPath 2.0). To resolve the references from the files returned by the `collection()` function with an XML catalog, specify the class name of the XML catalog enabled parser for parsing these collection files. The class name is `ro.sync.xml.parser.CatalogEnabledXMLReader`. Specify it as it follows:

```
let $docs := collection(iri-to-uri(
 "file:///D:/temp/test/XQuery-catalog/mydocsdir?reurse=yes;select=*.xml;
 parser=ro.sync.xml.parser.CatalogEnabledXMLReader"))
```

## XPath Prefix Mapping

To define default mappings between prefixes (that you can use in the XPath toolbar) and namespace URIs [go to XQuery Options preferences panel](#) and enter the mappings in the **Default prefix-namespace mappings** table. The same preferences panel allows you to configure the default namespace used in XPath 2.0 expressions.



**Important:** If you define a default namespace, Oxygen XML Editor binds this namespace to the first free prefix from the list: `default`, `default1`, `default2`, and so on. For example, if you define the default namespace `xmlns="something"` and the prefix `default` is not associated with another namespace, you can match tags without prefix in an XPath expression typed in the XPath toolbar by using the prefix `default`. To find all the `level` elements when you define a default namespace in the root element, execute this expression: `//default:level` in the XPath toolbar.

## Working with XQuery

This section explains how to edit and run XQuery queries in Oxygen XML Editor.

### What is XQuery

XQuery is the query language for XML and is officially defined by [a W3C Recommendation document](#). The many benefits of XQuery include:

- XQuery allows you to work in one common model no matter what type of data you're working with: relational, XML, or object data.
- XQuery is ideal for queries that must represent results as XML, to query XML stored inside or outside the database, and to span relational and XML sources.
- XQuery allows you to create many different types of XML representations of the same data.
- XQuery allows you to query both relational sources and XML sources, and create one XML result.

### Syntax Highlight and Content Completion

To [create an XQuery document](#), select **File > New (Ctrl (Meta on Mac OS)+N)** and when the **New** dialog appears select XQuery entry.

Oxygen XML Editor provides syntax highlight for keywords and all known XQuery functions and operators. A content completion assistant is also available and can be activated with the **(Ctrl (Meta on Mac OS)+Space)** shortcut. The functions and operators are presented together with a description of the parameters and functionality. For some supported database engines like eXist and Berkeley DB, the content completion list offers the specific XQuery functions implemented by that engine. This feature is available when the XQuery file has an associated transformation scenario which uses one of these database engines or the XQuery validation engine is set to one of them via a validation scenario or in the [XQuery Preferences](#) page.

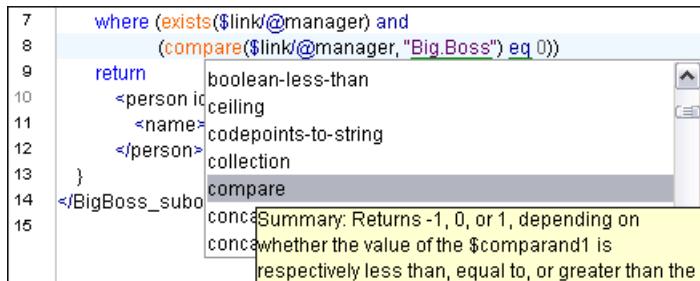
The extension functions built in the Saxon product are available on content completion if one of the following conditions are true:

- the edited file has a transformation scenario associated that uses as transformation engine Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE
- the edited file has a validation scenario associated that use as validation engine Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE
- the validation engine specified in *Preferences* is Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE.

If the Saxon namespace (<http://saxon.sf.net>) is mapped to a prefix the functions are presented using this prefix, the default prefix for the Saxon namespace (saxon) is used otherwise.

If you want to use a function from a namespace mapped to a prefix, just type that prefix and the content completion displays all the XQuery functions from that namespace. When the default namespace is mapped to a prefix the XQuery functions from this namespace offered by content completion are also prefixed, only the function name being used otherwise.

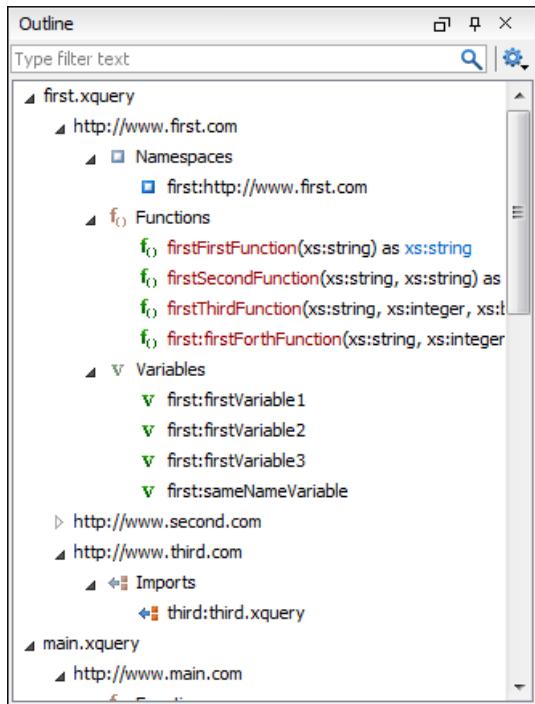
The content completion popup window presents all the variables and functions from both the edited XQuery file and its imports.



**Figure 295: XQuery Content Completion**

## XQuery Outline View

The XQuery document structure is presented in the **XQuery Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to a component by knowing its name. It can be opened from the **Window > Show View > Outline** menu action.



**Figure 296: XQuery Outline View**

The following actions are available in the **Settings** menu on the Outline view's toolbar:

**Selection update on caret move**

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret moves or the changes performed in the XQuery editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

**Sort**

Allows you to alphabetically sort the XQuery components.

**Show all components**

Displays all collected components starting from the current file. This option is set by default.

**Show only local components**

Displays the components defined in the current file only.

**Group by location/namespace/type**

Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the **Outline** view.

If you know the component name, you can search it in the **Outline** view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, (**Enter**), (**Tab**), (**Shift-Tab**). To switch from tree structure to the filter text field, you can use (**Tab**), (**Shift-Tab**).

**Tip:** The search filter is case insensitive. The following wildcards are accepted:

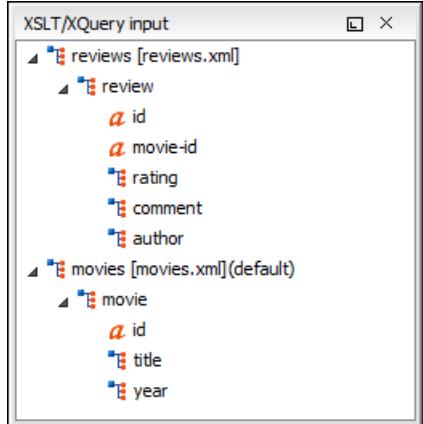
- \* - any string
- ? - any character
- , - patterns separator

If no wildcards are specified, the string to search is used as a partial match (like `*textToFind*`).

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (\*, ?) and separate multiple patterns with commas.

## The XQuery Input View

You are able to drag and drop a node in the editing area to insert XQuery expressions quickly.



**Figure 297: XQuery Input view**

### Create FLWOR by drag and drop

For the following XML documents:

```
<movies>
<movie id="1">
<title>The Green Mile</title>
<year>1999</year>
</movie>
<movie id="2">
<title>Taxi Driver</title>
<year>1976</year>
</movie>
</movies>
```

and

```
<reviews>
<review id="100" movie-id="1">
<rating>5</rating>
<comment>It is made after a great Stephen King book.</comment>
<author>Paul</author>
</review>
<review id="101" movie-id="1">
<rating>3</rating>
<comment>Tom Hanks does a really nice acting.</comment>
<author>Beatrice</author>
</review>
<review id="104" movie-id="2">
<rating>4</rating>
<comment>Robert De Niro is my favorite actor.</comment>
<author>Maria</author>
</review>
</reviews>
```

and the following XQuery:

```
let $review := doc("reviews.xml")
for $movie in doc("movies.xml")/movies/movie
let $movie-id := $movie/@id
return
<movie id="{{$movie/@id}}>
```

```

{$movie/title}
{$movie/year}
<maxRating>
{
}
</maxRating>
</movie>
```

if you drag the **review** element and drop it between the braces a popup menu will be displayed.



**Figure 298: XQuery Input drag and drop popup menu**

Select **FLWOR rating** and the result document will be:

```

37 {
38 for $review in doc("reviews.xml")/reviews/review
39 return
40 where ((compare($review/rating/text(), string($minRating)) eq 0)
41 and ($review/@movie-id = $movie/@id))
42 return $review/author
43 }
44 </minRating>
```

**Figure 299: XQuery Input drag and drop result**

## XQuery Validation

With Oxygen XML Editor, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.6.0.5 PE processor or the 9.6.0.5 EE, IBM DB2, eXist, Berkeley DB XML, Documentum xDb (X-Hive/DB) 10, or MarkLogic (version 5 or newer) if you installed them. Any other XQuery processor that offers an [XQJ API implementation](#) can also be used. This is in conformance with [the XQuery Working Draft](#). The processor is used in two cases: validation of the expression and execution. Although the execution implies a validation, it is faster to check the expression syntactically, without executing it. The errors that occurred in the document are presented in the messages view at the bottom of editor window, with a full description message. As with all error messages, if you click one entry, the line where the error appeared is highlighted.

The screenshot shows the Oxygen XML Editor interface with a code editor window titled "personal.xquery\*". The editor displays an XQuery script with syntax highlighting. A status bar at the bottom indicates a static error: "F [Saxon-PE XQuery 9.2.1.2] XQuery static error in #...ink/@manager, 'Big.Boss') eq 0#: Unknown system function". Below the editor is a toolbar with several icons, including a validation icon. A message box is open, showing the error details: "Info Description - 1 item Resource Location" followed by the error message. The main menu bar at the top includes "File", "Edit", "View", "Tools", "Help", and "Oxygen XML Editor".

**Figure 300: XQuery Validation**

**Note:** In case you choose a processor that does not support XQuery validation, Oxygen XML Editor displays a warning when trying to validate.

The **Validation options** button, available in the **Document > Validate** menu, allows quick access to the [XQuery options](#) in the Oxygen XML Editor preferences.

When you open an XQuery document from a connection that supports validation (for example MarkLogic, or eXist), by default Oxygen XML Editor uses this connection for validation. In case you open an XQuery file using a MarkLogic connection, the validation better resolves imports.

## Other XQuery Editing Actions

The XQuery editor offers a reduced version of *the popup menu available in the XML editor type*:

- *the split actions*,
- *the folding actions*
- *the edit actions*
- a part of *the source actions*:
  - **To lower case**
  - **To upper case**
  - **Capitalize lines**
- open actions:
  - **Open file at Caret**
  - **Open file at Caret in System Application**

## Transforming XML Documents Using XQuery

XQueries are similar with the XSL stylesheets, both being capable of transforming an XML input into another format. You specify the input URL when you *define the transformation scenario*. The result can be saved and opened in the associated application. You can even run a *FO processor* on the output of an XQuery. The transformation scenarios may be shared between many XQuery files, are *exported* together with the XSLT scenarios and can be managed in *the Configure Transformation Scenario dialog*, or in *the Scenarios view*. The transformation can be performed on the XML document specified in the **XML URL** field, or, if this field is empty, the documents referenced from the query expression. The parameters of XQuery transforms must be set in *the Parameters dialog*. Parameters that are in a namespace must be specified using the qualified name, for example a param parameter in the <http://www.oxygenxml.com/ns> namespace must be set with the name {http://www.oxygenxml.com/ns}param.

The transformation uses one of the Saxon 9.6.0.5 HE, Saxon 9.6.0.5 PE, Saxon 9.6.0.5 EE processors, a database connection (details can be found in the [Working with Databases](#) chapter - in the [XQuery transformation](#) section) or any XQuery processor that provides an XQJ API implementation.

The Saxon 9.6.0.5 EE processor supports also XQuery 3.0 transformations.

## XQJ Transformers

This section describes the necessary procedures before running an XQJ transformation.

### How to Configure an XQJ Data Source

Any transformer that offers an XQJ API implementation can be used when validating XQuery or transforming XML documents. An example of an XQuery engine that implements the XQJ API is [Zorba](#).

1. In case your XQJ Implementation is native, make sure the directory containing the native libraries of the engine is added to your system environment variables: to PATH - on Windows, to LD\_LIBRARY\_PATH - on Linux, or to DYLD\_LIBRARY\_PATH - on OS X. Restart Oxygen XML Editor after configuring the environment variables.
2. [Open the Preferences dialog box](#) and go to **Data Sources**.
3. Click the **New** button in the **Data Sources** panel.
4. Enter a unique name for the data source.
5. Select **XQuery API for Java(XQJ)** in the **Type** combo box.
6. Press the **Add** button to add XQJ API-specific files.

You can manage the driver files using the **Add**, **Remove**, **Detect**, and **Stop** buttons.

Oxygen XML Editor detects any implementation of `javax.xml.xquery.XQDataSource` and presents it in **Driver class** field.

7. Select the most suited driver in the **Driver class** combo box.
8. Click the **OK** button to finish the data source configuration.

### How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for this connection.
4. Select one of the previously configured [XQJ data sources](#) in the **Data Source** combo box.
5. Fill-in the connection details.

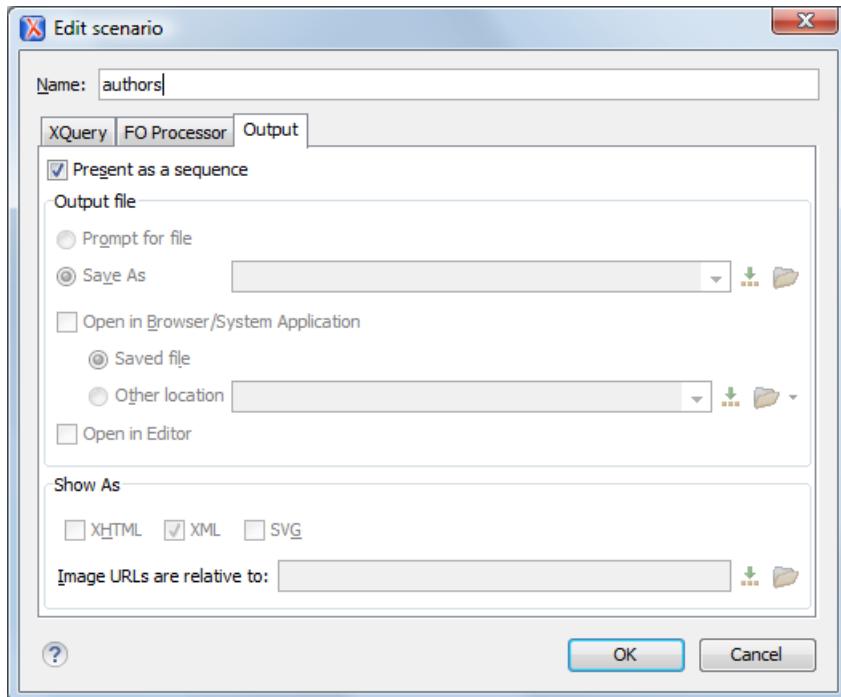
The properties presented in the connection details table are automatically detected depending on the selected data source.

6. Click the **OK** button.

### Display Result in Sequence View

The result of an XQuery executed on a database can be very large and sometimes only a part of the full result is needed. To avoid the long time necessary for fetching the full result, select the **Present as a sequence** option in the **Output** tab of the **Edit scenario** dialog. This option fetches only the first chunk of the result. Clicking the **More results available** label that is displayed at the bottom of the **Sequence** view fetches the next chunk of results.

The size of a chunk can be [set with the option Size limit of Sequence view](#). The  **XQuery options** button from the **More results available** label provides a quick access to this option by opening [the XQuery panel of the Preferences dialog](#) where the option can be modified.



**Figure 301:** The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the Sequence view.

```

<ns> 42900. element
{N} 42989. element
{N} 42990. element
{N} 42991. element
{N} 42992. element
{N} 42993. element
{N} 42994. element
{N} 42995. element
{N} 42996. element
{N} 42997. element
{N} 42998. element
{N} 42999. element
{N} 43000. element

```

**Figure 302:** The XQuery transformation result displayed in Sequence view

#### Advanced Saxon HE/PE/EE XQuery Transformation Options

The XQuery transformation scenario allows you to configure advanced options that are specific for the Saxon HE (Home Edition), PE (Professional Edition), and EE (Enterprise Edition) engines. They are the same options as [the values set in the user preferences](#) but they are configured as a specific set of transformation options for each transformation scenario.

The advanced options for Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

- **Recoverable errors ("‐warnings")**- Allows the user to choose how dynamic errors are handled. The following options can be selected:
  - **recover silently ("silent")** - Continues processing without reporting the error.
  - **recover with warnings ("recover")** - Issues a warning but continues processing.
  - **signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.
- **Strip whitespaces ("‐strip")** - Can have one of the following values:
  - **All ("all")** - Strips *all* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document.

- **Ignore ("ignorable")** - Strips all *ignorable* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
- **None ("none")** - Strips *no* whitespace before further processing.
- **Optimization level ("-opt")** - This option allows optimization to be suppressed in cases where reducing the compiling time is important, where optimization conflicts with debugging, or causes extension functions with side-effects to behave unpredictably.
- **Use linked tree model ("-tree:linked")** - This option activates the linked tree model.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - If checked, Saxon runs the XQuery transformation with the XQuery 3.0 support (this option is enabled by default).
- **Initializer class** - Equivalent to the `-init` Saxon command-line argument. The value is the name of a user-supplied class that implements the `net.sf.saxon.lib.Initializer` interface. This initializer is called during the initialization process, and may be used to set any options required on the configuration programmatically. It is particularly useful for tasks such as registering extension functions, collations, or external object models, especially in Saxon-HE where the option cannot be set via a configuration file. Saxon only calls the initializer when running from the command line, but the same code may be invoked to perform initialization when running user application code.



**Important:** The *advanced Saxon-HE/PE/EE options configured in a transformation scenario* override the Saxon-HE/PE/EE options defined globally.

The following advanced options are specific for Saxon 9.6.0.5 Professional Edition (PE) and Enterprise Edition (EE) only:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that is used for XQuery transformation and validation
- **Allow calls on extension functions ("-ext")** - If checked, calls on external functions are allowed. Checking this option is recommended in an environment where untrusted stylesheets may be executed. It also disables user-defined extension elements and the writing of multiple output files, both of which carry similar security risks.

The advanced options that are specific for Saxon 9.6.0.5 Enterprise Edition (EE) are as follows:

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and enables parsing the source documents with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Generate bytecode ("--generateByteCode:(on|off)")** - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to <http://www.saxonica.com/documentation9.5/index.html#!javadoc>.
- **Enable XQuery update ("-update:(on|off)")** - This option controls whether or not XQuery update syntax is accepted.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, backup versions for any XML files updated with XQuery Update are generated. This option is available when the **Enable XQuery update** option is enabled.

## Updating XML Documents using XQuery

Using the bundled Saxon 9.6.0.5 EE XQuery processor Oxygen XML Editor offers support for XQuery Update 1.0. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model. Thus, besides querying XML documents, you can modify them using the various insert/delete/modify/create methods available in the [XQuery Update 1.0](#) standard.

Choose Saxon 9.6.0.5 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Editor will notify you if the update was successful.

### Using XQuery Update to modify a tag name in an XML file

```
rename node doc("books.xml")//publisher[1]//book[1] as "firstBook"
```

---

# Chapter

# 13

---

## Debugging XSLT Stylesheets and XQuery Documents

---

**Topics:**

- [Overview](#)
- [Layout](#)
- [Working with the XSLT / XQuery Debugger](#)
- [Debugging Java Extensions](#)
- [Supported Processors for XSLT / XQuery Debugging](#)

This chapter explains the user interface and how to use the debugger with XSLT and XQuery transformations.

## Overview

---

The **XSLT Debugger** and **XQuery Debugger** perspectives enable you to test and debug XSLT 1.0 / 2.0 / 3.0 stylesheets and XQuery 1.0 / 3.0 documents including complex XPath 2.0 / 3.0 expressions. The interface presents simultaneous views of the source XML document, the XSLT/XQuery document and the result document. As you go step by step through the XSLT/XQuery document the corresponding output is generated step by step, and the corresponding position in the XML file is highlighted. At the same time, special views provide various types of debugging information and events useful to understand the transformation process.

The following set of features allow you to test and solve XSLT/XQuery problems:

- Support for XSLT 1.0 stylesheets (using Saxon 6.5.5 and Xalan XSLT engines), XSLT 2.0 / 3.0 stylesheets and XPath 2.0 / 3.0 expressions that are included in the stylesheets (using Saxon 9.6.0.5 XSLT engine) and XQuery 1.0 / 3.0 (using Saxon 9.6.0.5 XQuery engine).
- Stepping capabilities: step in, step over, step out, run, run to cursor, run to end, pause, stop.
- Output to source mapping between every line of output and the instruction element / source context that generated it.
- Breakpoints on both source and XSLT/XQuery documents.
- Call stack on both source and XSLT/XQuery documents.
- Trace history on both source and XSLT/XQuery documents.
- Support for XPath expression evaluation during debugging.
- Step into imported/include stylesheets as well as included source entities.
- Available templates and hits count.
- Variables view.
- Dynamic output generation.

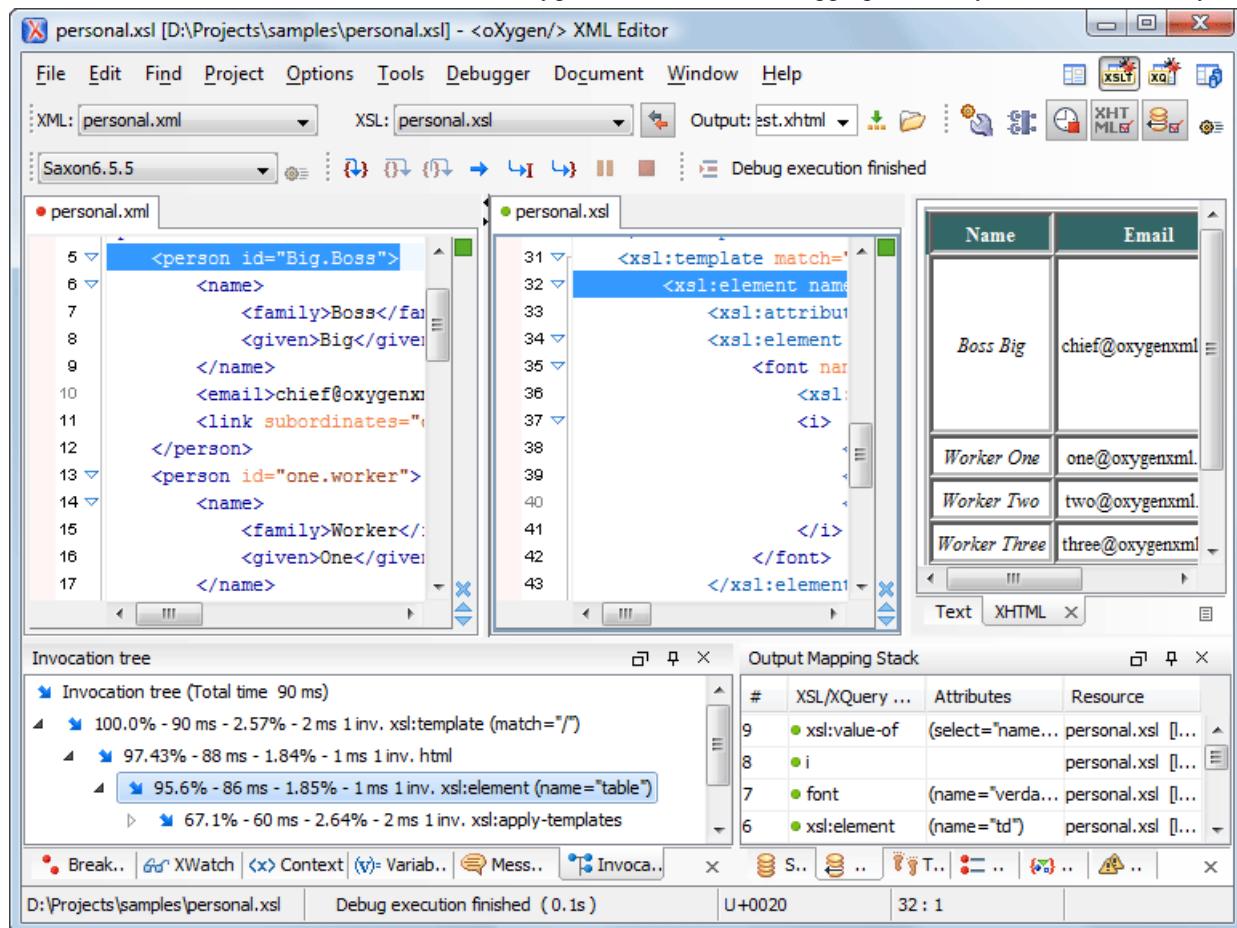
## Layout

---

The XML and XSL files are displayed in *Text mode*. The other modes (*Author mode*, *Grid mode*) are available only in *the Editor perspective*.

The debugger perspective contains four sections:

- **Source document view (XML)** - Displays and allows the editing of XML files (documents).
- **XSLT/XQuery document view (XSLT/XQuery)** - Displays and allows the editing of XSL files(stylesheets) or XQuery documents.
- **Output document view** - Displays the output that results from inputting a document (XML) and a stylesheet (XSL) or XQuery document in the transformer. The transformation result is written dynamically while the transformation is processed. There are two types of output views: a text view (with XML syntax highlight) and an XHTML view. For large outputs, the XHTML view can be disabled (see *Debugger Settings*).
- **Control view** - The control view is used to configure and control the debugging operations. It also provides a set of *Information views* types. This pane has two sections:
  - *Control toolbar*
  - *Information views*



**Figure 303: Debugger Mode Interface**

XML documents and XSL stylesheets or XQuery documents that were opened in the Editor perspective are automatically sorted into the first two panes. When multiple files of each type are opened, the individual documents and stylesheets are separated using the familiar tab management system of the Editor perspective. Selecting a tab brings the document or stylesheet into focus and enables editing without the need to go back to the Editor perspective.

When editing in the Editor perspective the editor toolbar is displayed. In Debugger mode this toolbar is not available, however the functions are still accessible from [the Document menu](#) and [the editors contextual menus](#).

Bookmarks are replaced in the Debugger perspective by breakpoints.

During debugging, the current execution node is highlighted in both document (XML) and XSLT/XQuery views.

## Control Toolbar

The **Control** toolbar contains all the actions that you need to configure and control the debugging process. The following actions are described as they appear in the toolbar from left to right.



**Figure 304: Control Toolbar**

### XML source selector

The current selection represents the source document used as input by the transformation engine. A drop-down list contains all opened files (XML files being emphasized). This option allows you to use other file types also as source

documents. In an XQuery debugging session this selection field can be set to the default value NONE, because usually XQuery documents do not require an input source.

### XSL / XQuery selector

The current selection represents the stylesheet or XQuery document to be used by the transformation engine. The selection list contains all opened files (XSLT / XQuery files being emphasized).

#### Link with editor

When enabled, the XML and XSLT/XQuery selectors display the names of the files opened in the central editor panels. This button is disabled by default.

### Output selector

The selection represents the output file specified in the associated transformation scenario.

#### XSLT / XQuery parameters

XSLT / XQuery parameters to be used by the transformation.

#### Libraries

Add and remove the Java classes and jars used as XSLT extensions.

#### Turn on profiling

Enables / Disables current transformation profiling.

#### Enable XHTML output

Enables the rendering of the output in the XHTML output view during the transformation process. For performance issues, disable XHTML output when working with very large files. Note that only XHTML conformant documents can be rendered by this view. In order to view the output result of other formats, such as HTML, save the **Text output** area to a file and use an external browser for viewing.

When starting a debug session from the editor perspective using the **Debug Scenario** action, the state of this toolbar button reflects the state of the **Show as XHTML** output option from the scenario.

#### Turn on/off output to source mapping

Enables or disables the output to source mapping between every line of output and the instruction element / source context that generated it.

#### Debugger preferences

Quick link to [Debugger preferences page](#).

### XSLT / XQuery engine selector

Lists the [processors available for debugging XSLT and XQuery transformations](#).

#### XSLT / XQuery engine advanced options

Advanced options available for Saxon 9.6.0.5.

#### Step into

Starts the debugging process and runs until the next instruction is encountered.

#### Step over

Run until the current instruction and its sub-instructions are over. Usually this will advance to the next sibling instruction.

The screenshot shows an XSLT code editor with line numbers 12 to 20. A blue box highlights line 13: <h3 style="color:blue">. A red arrow points from this line down to line 19: <xsl:message>Step over goes here</xsl:message>. A small red icon with a question mark is located at the end of the red arrow.

```

12 <xsl:template match="CCC" priority="4">
13 <h3 style="color:blue"> ↗
14 <xsl:value-of select="name0"/>
15 <xsl:text> (id=</xsl:text>
16 <xsl:value-of select="@id"/>
17 <xsl:text>)</xsl:text>
18 </h3>
19 <xsl:message>Step over goes here</xsl:message>
20 </xsl:template>

```

**Figure 305: Step over**

### Step over

Run until the parent of the current instruction is over. Usually this will advance to the next sibling of the parent instruction.

The screenshot shows the same XSLT code as Figure 305. A blue box highlights line 13: <h3 style="color:blue">. A red arrow points from this line down to line 19: <xsl:message>Step out goes here</xsl:message>. A small red icon with a question mark is located at the end of the red arrow.

```

12 <xsl:template match="CCC" priority="4">
13 <h3 style="color:blue"> ↗
14 <xsl:value-of select="name0"/>
15 <xsl:text> (id=</xsl:text>
16 <xsl:value-of select="@id"/>
17 <xsl:text>)</xsl:text>
18 </h3>
19 <xsl:message>Step out goes here</xsl:message>
20 </xsl:template>

```

**Figure 306: Step out**

### Run

Starts the debugging process. The execution of the process is paused when a *breakpoint* is encountered or the transformation ends.

### Run to cursor

Starts the debugging process and runs until one of the following conditions occur: the line of cursor is reached, a valid breakpoint is reached or the execution ends.

### Run to end

Runs the transformation until the end, without taking into account enabled breakpoints, if any.

### Pause

Request to pause the current transformation as soon as possible.

### Stop

Request to stop the current transformation without completing its execution.

### Show current execution nodes

Reveals the current debugger context showing both the current instruction and the current node in the XML source. Possible displayed states:

- entering ( ) or leaving ( ) an XML execution node
- entering ( ) or leaving ( ) an XSL execution node
- entering ( ) or leaving ( ) an XPath execution node



**Note:** When you set a MarkLogic server as a processor, the **Show current execution nodes** button is named **Refresh current session context from server**. Click this button to refresh the information in all the views.



**Note:** For some of the XSLT processors (Saxon-HE/PE/EE) the debugger could be configured to step into the XPath expressions affecting the behavior of the following debugger actions: **Step into**, **Step over** or **Step Out**.

## Information View

The **Information** view is comprised of two panes that are used to display various types of information used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This enables the developer to obtain a clear view of the transformation progress. Using the debug controls developers can easily isolate parts of stylesheet therefore they may be understood and modified. The information types include:

Left side information views

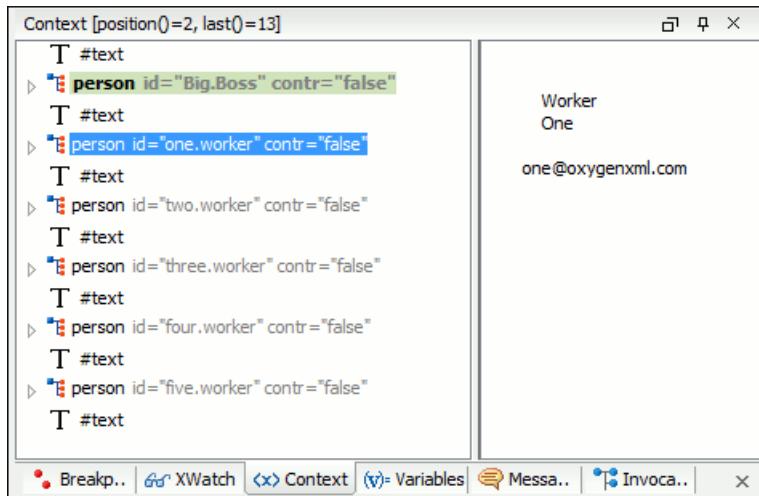
- [Context Node view](#)
- [XWatch view](#)
- [Breakpoints view](#)
- [Messages view](#) (XSLT only)
- [Variables view](#)

Right side information views

- [Stack view](#)
- [Trace view](#)
- [Templates view](#) (XSLT only)
- [Nodeset view](#)

### Context Node View

The context node is valid only for XSLT debugging sessions and is a source node corresponding to the XSL expression that is evaluated. It is also called the context of execution. The context node implicitly changes as the processor hits various steps (at the point where XPath expressions are evaluated). This node has the same value as evaluating '.' (dot) XPath expression in [XWatch view](#). The value of the context node is presented as a tree in the view.



**Figure 307: The Context node view**

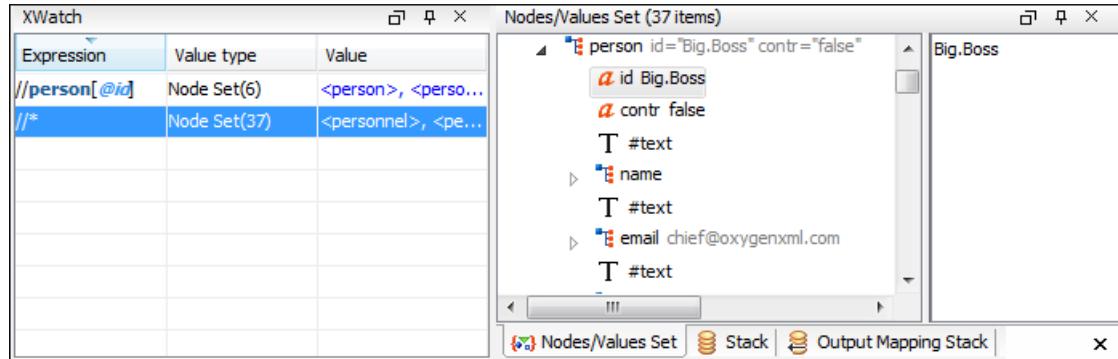
The context node is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix, the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel. The **Context** view also presents the current mode of the XSLT processor in case this mode differs from the default one.

The title bar displays the current element index and the number of elements that compose the current context (this information is not available if you choose Xalan or Saxon 6 as processing engine).

## XPath Watch (XWatch) View

The **XWatch** view shows XPath expressions evaluated during the debugging process. Expressions are evaluated dynamically as the processor changes its source context.

When you type an XPath expression in the **Expression** column, Oxygen XML Editor supports you with syntax highlight and *content completion assistance*.



**Figure 308: The XPath watch view**

**Table 10: XWatch columns**

Column	Description
Expression	XPath expression to be evaluated (XPath 1.0 or 2.0 / 3.0 compliant).
Value	Result of XPath expression evaluation. Value has a type (see <i>the possible values</i> in the section <b>Variables View</b> on page 779). For <i>Node Set</i> results, the number of nodes in the set is shown in parenthesis.

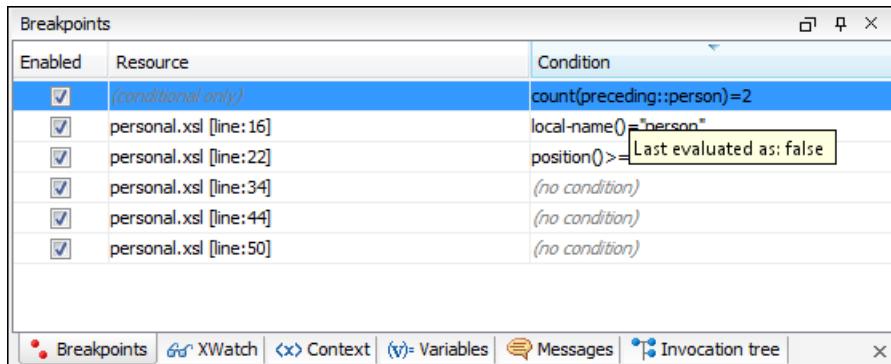


**Important:** Remarks about working with the **XWatch** view:

- Expressions that reference variable names are not evaluated.
- The expression list is not deleted at the end of the transformation (it is preserved between debugging sessions).
- To insert a new expression, click the first empty line of the **Expression** column and start typing. As an alternative, right click and select the **Add** action. Press **(Enter)** on the cell to add and evaluate.
- To delete an expression, click on its **Expression** column and delete its content. As an alternative, right click and select the **Remove** action. Press **(Enter)** on the cell to commit changes.
- If the expression result type is a *Node Set*, click it (**Value** column) and its value is displayed in the **Nodes/Values Set view**.
- The **Copy**, **Add**, **Remove** and **Remove All** actions are available in every row's contextual menu.

## Breakpoints View

This view lists all breakpoints that are set on opened documents. Once you *insert a breakpoint* it is automatically added in this list. Breakpoints can be set in XSLT/XQuery documents and XML documents in XSLT/XQuery debugging sessions. A breakpoint can have an associated break condition that represents an XPath expression evaluated in the current debugger context. In order to be processed, their evaluation result should be a boolean value. A breakpoint with an associated condition only stops the execution of the Debugger if the breakpoint condition is evaluated as **true**.



**Figure 309: The Breakpoints View**

The **Breakpoints** view contains the following columns:

- **Enabled** - If checked, the current condition is evaluated and taken into account.
- **Resource** - Resource file and number of the line where the breakpoint is set. The Entire path of resource file is available as tooltip.
- **Condition** - XSLT/XQuery expression to be evaluated during debugging. The expression will be evaluated at every debug step.

Clicking a record highlights the breakpoint line in the document.

**Note:** The breakpoints list is not deleted at the end of a transformation (it is preserved between debugging sessions).

The following actions are available in the contextual menu of the table:

#### Go to

Moves the cursor to the source of the breakpoint.

#### Enable

Enables the breakpoint.

#### Disable

Disables the breakpoint. A disabled breakpoint will not be evaluated by the Debugger.

#### Add

Allows you to add a new breakpoint and breakpoint condition.

#### Edit

Allows you to edit an existing breakpoint.

#### Remove

Deletes the selected breakpoint.

#### Enable all

Enables all breakpoints.

#### Disable all

Disables all breakpoints.

#### Remove all

Removes all breakpoints.

### Messages View

`xsl:message` instructions are one way to signal special situations encountered during transformation as well as a raw way of doing the debugging. This view is available only for XSLT debugging sessions and shows all `xsl:message` calls executed by the XSLT processor during transformation.

Messages		
Message	Terminate	Resource
Message 1	no	personal.xsl [line: 8]
Message 2	no	personal.xsl [line: 12]
Message 3	no	personal.xsl [line: 29]

**Figure 310: The Messages View****Table 11: Messages columns**

Column	Description
Message	Message content.
Terminate	Signals if processor terminates the transformation or not once it encounters the message (yes/no respectively).
Resource	Resource file where <code>xsl:message</code> instruction is defined and the message line number. The complete path of the resource is available as tooltip.

The following actions are available in the contextual menu:

#### Go to

Highlight the XSL fragment that generated the message.

#### Copy

Copies to clipboard message details (system ID, severity info, description, start location, terminate state).



#### Important: Remarks

- Clicking a record from the table highlights the `xsl:message` declaration line.
- Message table values can be sorted by clicking the corresponding column header. Clicking the column header switches the sorting order between: ascending, descending, no sort.

## Stack View

This view shows the current execution stack of both source and XSLT/XQuery nodes. During transformation two stacks are managed: one of source nodes being processed and the other for XSLT/XQuery nodes being processed. Oxygen XML Editor shows both node types into one common stack. The source (XML) nodes are preceded by a red color icon while XSLT/XQuery nodes are preceded by a green color icon. The advantage of this approach is that you can always see the source scope on which a XSLT/XQuery instruction is executed (the last red color node on the stack). The stack is oriented upside down.

Stack			
#	XML/XSL/XQuery Node	Attributes	Resource
4	green xsl:message		personal.xsl
3	green xsl:element	(name = "table")	personal.xsl
2	green html		personal.xsl
1	green xsl:template	(match = "/")	personal.xsl
0	red #document		personal.xml

Stack Trace Templates Nodes/Val. Hotspots Results

**Figure 311: The Stack View**

The contextual menu contains one action: **Go to**, which moves the selection in the editor panel to the line containing the XSLT element that is displayed on the selected line from the view.

**Table 12: Stack columns**

Column	Description
#	Order number, represents the depth of the node (0 is the stack base).
XML/XSLT/XQuery Node	Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as <b>#document</b> .
Attributes	Attributes of the node (a list of <code>id="value"</code> pairs).
Resource	Resource file where the node is located. The entire path is available as tooltip.

**Important:** Remarks:

- Clicking a record from the stack highlights that node's location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: `xsl:template` using Saxon and `template` using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

**Output Mapping Stack View**

The **Output Mapping Stack** view displays *context data* and presents the XSLT templates/XQuery elements that generated specific areas of the output.

Output Mapping Stack			
#	XML/XSL/XQuery Node	Attributes	Resource
8	xsl:value-of	(select=".//link/@manager")	personal.xsl
7	font	(color="black") (name="verdana") (size="3")	personal.xsl
6	xsl:element	(name="td")	personal.xsl
5	xsl:element	(name="tr")	personal.xsl
4	xsl:template	(match="//person")	personal.xsl
3	xsl:apply-templates		personal.xsl
2	xsl:element	(name="table")	personal.xsl
1	html		personal.xsl
0	xsl:template	(match="/")	personal.xsl

Stack   Output Mapping Stack   Trace   Templates   Nodes/Values Set   X

**Figure 312: The Output Mapping Stack view**

The **Go to** action of the contextual menu takes you in the editor panel at the line containing the XSLT element displayed in the **Output Mapping Stack** view.

**Table 13: Output Mapping Stack columns**

Column	Description
#	The order number in the stack of XSLT templates/XQuery elements. Number 0 corresponds to the bottom of the stack in the status of the XSLT/XQuery processor. The highest number corresponds to the top of the stack.
XSL/XQuery Node	The name of an XSLT template/XQuery element that participated in the generation of the selected output area.
Attributes	The attributes of the XSLT template/XQuery node.

Column	Description
Resource	The name of the file containing the XSLT template/XQuery element.



### Important: Remarks:

- Clicking a record highlights that XSLT template definition/XQuery element inside the resource (XSLT stylesheet file/XQuery file);
- Saxon only shows the applied XSLT templates having at least one hit from the processor. Xalan shows all defined XSLT templates, with or without hits;
- The table can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort;
- Xalan shows also the built-in XSLT rules.

## Trace History View

Usually the XSLT/XQuery processors signal the following events during transformation:

- → - Entering a source (XML) node.
- ← - Leaving a source (XML) node.
- ➤ - Entering a XSLT/XQuery node.
- ➛ - Leaving a XSLT/XQuery node.

The trace history catches all these events, so you can see how the process evolved. The red icon lines denote source nodes while the green icon lines denote XSLT/XQuery nodes.

It is possible to save the element trace in a structured XML document. The action is available on the context menu of the view. In this way you have the possibility to compare the trace results from different debug sessions.

Trace			
Depth	XML/XSL/XQuery Node	Attributes	Resource
0	→ #document		personal.xml
1	➤ xsl:template	(match="/")	personal.xsl
2	➤ html		personal.xsl
3	➤ xsl:element	(name="table")	personal.xsl
4	➤ xsl:message		personal.xsl
4	➤ xsl:message		personal.xsl
4	➤ xsl:message		personal.xsl

Stack Trace Templates Nodes/Values Set Hotspots Results

**Figure 313: The Trace History View**

The contextual menu contains the following actions:

### Go to

Moves the selection in the editor panel to the line containing the XSLT element or XML element that is displayed on the selected line from the view;

### Export to XML

Saves the entire trace list into XML format.

**Table 14: Trace History columns**

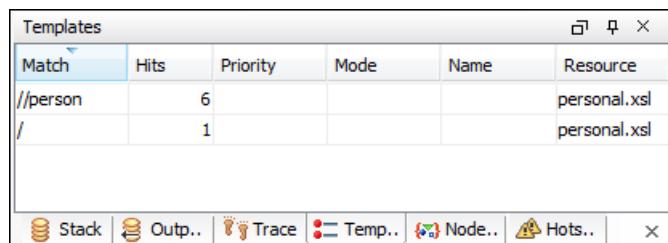
Column	Description
Depth	Shows you how deep the node is nested in the XML or stylesheet structure. The bigger the number, the more nested the node is. A depth 0 node is the document root.
XML/XSLT/XQuery Node	Represents the node from the processed source or stylesheet document. One particular node is the document root, noted as #document. Every node is preceded by an arrow that represents what action was performed on it (entering or leaving the node).
Attributes	Attributes of the node (a list of id="value" pairs).
Resource	Resource file where the node is located. The complete path of the resource file is provided as tooltip.

**Important:** Remarks:

- Clicking a record highlights that node's location inside the resource.
- Only the Saxon processor shows the element attributes.
- The Xalan processor shows also the built-in rules.

**Templates View**

The `xsl:template` is the basic element for stylesheets transformation. This view is only available during XSLT debugging sessions and shows all `xsl:template` instructions used by the transformation. By seeing the number of hits for each of the templates you get an idea of the stylesheet coverage by template rules with respect to the input source.

**Figure 314: The Templates view**

The contextual menu contains one action: **Go to**, which moves the selection in the editor panel to the line containing the XSLT template that is displayed on the selected line from the view.

**Table 15: Templates columns**

Column	Description
Match	The match attribute of the <code>xsl:template</code> .
Hits	The number of hits for the <code>xsl:template</code> . Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The mode attribute of the <code>xsl:template</code> .
Name	The name attribute of the <code>xsl:template</code> .

Column	Description
Resource	The resource file where the template is located. The complete path of the resource file is available as tooltip.



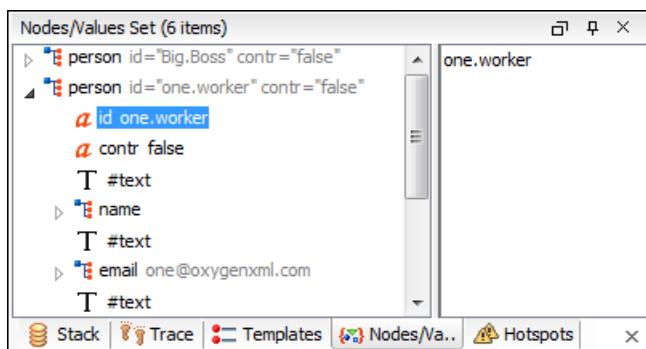
### Important: Remarks:

- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

## Nodes/Values Set View

This view is always used in relation with [The Variables view](#) and [the XWatch view](#). It shows an XSLT node set value in a tree form. The node set view is updated as response to the following events:

- You click a variable having a node set value in one of the above 2 views.
- You click a tree fragment in one of the above 2 views.
- You click an XPath expression evaluated to a node set in one of the above 2 views.



**Figure 315: The Node Set view**

The nodes / values set is presented in a tree-like fashion. The total number of items is presented in the title bar. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel.



### Important: Remarks:

- In case of longer values in the right side panel the interface shows three suspension points (...) at the end. A more detailed value is available as tooltip.
- Clicking a record highlights the location of that node into the source or stylesheet view.

## Variables View

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Editor uses the following icons to differentiate variables and parameters:

- - Global variable.
- - Local variable.
- - Global parameter.
- - Local parameter.

The following value types are available:

- **Boolean**
- **String**
- **Date** - XSLT 2.0 / 3.0 only.
- **Number**
- **Set**
- **Object**
- **Fragment** - Tree fragment.
- **Any**
- **Undefined** - The value was not yet set, or it is not accessible.



#### Note:

When Saxon 6.5 is used, if the value is unavailable, then the following message is displayed in the **Value** field: "The variable value is unavailable".

When Saxon 9 is used:

- If the variable is not used, the **Value** field displays "The variable is declared but never used".
- If the variable value cannot be evaluated, the **Value** field displays "The variable value is unavailable".

- **Document**
- **Element**
- **Attribute**
- **ProcessingInstruction**
- **Comment**
- **Text**
- **Namespace**
- **Evaluating** - Value under evaluation.
- **Not Known** - Unknown types.

Variables			
Variable/Parameter name filter <input type="text"/>			
	Name	Value type	Value
{V}	val	String	
{P}	rowval	String	1,2
V	trans	String	
P	level	String	1,2
P	image-path	String	Images/

Breakpoints | XWatch | Context | Variables | Messages | Invocation tree | X

Figure 316: The Variables View

Table 16: Variables columns

Column	Description
Name	Name of variable / parameter.
Value type	Type of variable/parameter.
Value	Current value of variable / parameter.

The value of a variable (the **Value** column) can be copied to the clipboard for pasting it to other editor area with the action **Copy value** from the contextual menu of the table from the view. This is useful in case of long and complex values which are not easy to remember by looking at them once.

**Important:** Remarks:

- Local variables and parameters are the first entries presented in the table.
- Clicking a record highlights the variable definition line.
- Variable values could differ depending on the transformation engine used or stylesheet version set.
- If the value of the variable is a node set or a tree fragment, clicking on it causes the ***Node Set view*** to be shown with the corresponding set of values.
- Variable table values can be sorted by clicking the corresponding column header. Clicking the column header switches between the orders: ascending, descending, no sort.

## Multiple Output Documents in XSLT 2.0 and XSLT 3.0

For XSLT 2.0 and XSLT 3.0 stylesheets that store the output in more than one file by using the `xsl:result-document` instruction the content of the file created in this way is displayed dynamically while the transformation is running in an output view. There is one view for each `xsl:result-document` instruction so that the output of different instructions is not mixed but is presented in different views.

## Working with the XSLT / XQuery Debugger

---

This section describes how to work with the debugger in the most common use cases.

To watch our video demonstration about how you can use the **XSLT Debugger**, go to [http://oxygenxml.com/demo/XSLT\\_Debugger.html](http://oxygenxml.com/demo/XSLT_Debugger.html).

### Steps in a Typical Debug Process

To debug a stylesheet or XQuery document follow the procedure:

1. [Open the source XML document](#) and [the XSLT/XQuery document](#).
2. If you are in the Editor perspective switch to the XSLT Debugger perspective or XQuery Debugger perspective with one of the actions (here explained for XSLT):
  - Menu **Window > Open perspective > XSLT Debugger** or the toolbar button XSLT Debugger
  - Menu **Document > XML Document > Debug Scenario** or the toolbar button Debug Scenario. This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.
3. Select the source XML document in the XML source selector of [the Control toolbar](#). In case of XQuery debugging if your XQuery document has no implicit source set the source selector value to **NONE**.
4. Select the XSLT/XQuery document in the XSLT/XQuery selector of [the Control toolbar](#).
5. Set XSLT/XQuery parameters from the button available on [the Control toolbar](#).
6. [Set one or more breakpoints](#).
7. Step through the stylesheet using the buttons available on [the Control toolbar](#):
  - Step into
  - Step over
  - Step out
  - Run
  - Run to cursor
  - Run to end
  - Pause
  - Stop

- Examine the information in the Information views to find the bug in the transformation process.

You may find [the procedure for determining the XSLT template/XQuery element that generated an output section](#) useful for fixing bugs in the transformation.

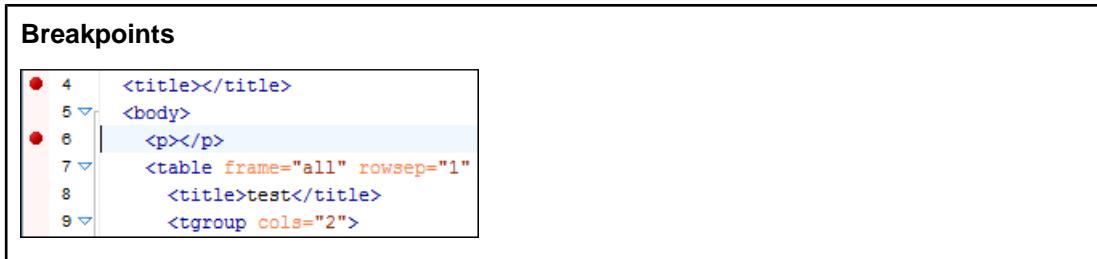
## Using Breakpoints

The Oxygen XML Editor XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure breakpoints are persistent between work sessions, they are saved at project level. You can set a maximum of 100 breakpoints per project.

### Inserting Breakpoints

To insert a breakpoint, follow these steps:

- Click the line where you want to insert the breakpoint in the XML source document or the XSLT/XQuery document.  
You can only set breakpoints on the XML source in XSLT or XQuery debugging sessions.  
Breakpoints are automatically created on the ending line of a start tag, even if you click a different line.
- Click the vertical stripe on the left side of the editor panel or select  **Create Breakpoint (Shift+F7)** from the **Edit > Breakpoints** menu.



### Removing Breakpoints

Only one action is required to remove a breakpoint:

Click the breakpoint icon in the vertical stripe on the left side of the editor panel or select **Remove all** from the **Edit > Breakpoints** menu.

## Determining What XSLT / XQuery Expression Generated Particular Output

In order to quickly spot the XSLT templates or XQuery expressions with problems it is important to know what XSLT template in the XSLT stylesheet or XQuery expression in the XQuery document and what element in the source XML document generated a specified area in the output.

Some of the debugging capabilities, for example *Step in* can be used for this purpose. Using *Step in* you can see how output is generated and link it with the XSLT/XQuery element being executed in the current source context. However, this can become difficult on complex XSLT stylesheets or XQuery documents that generate a large output.

You can click on the text from the **Text** output view or **XHTML** output view and the editor will select the XML source context and the XSLT template/XQuery element that generated the text. Also inspecting the whole stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the specified output area speeds up the debugging process.

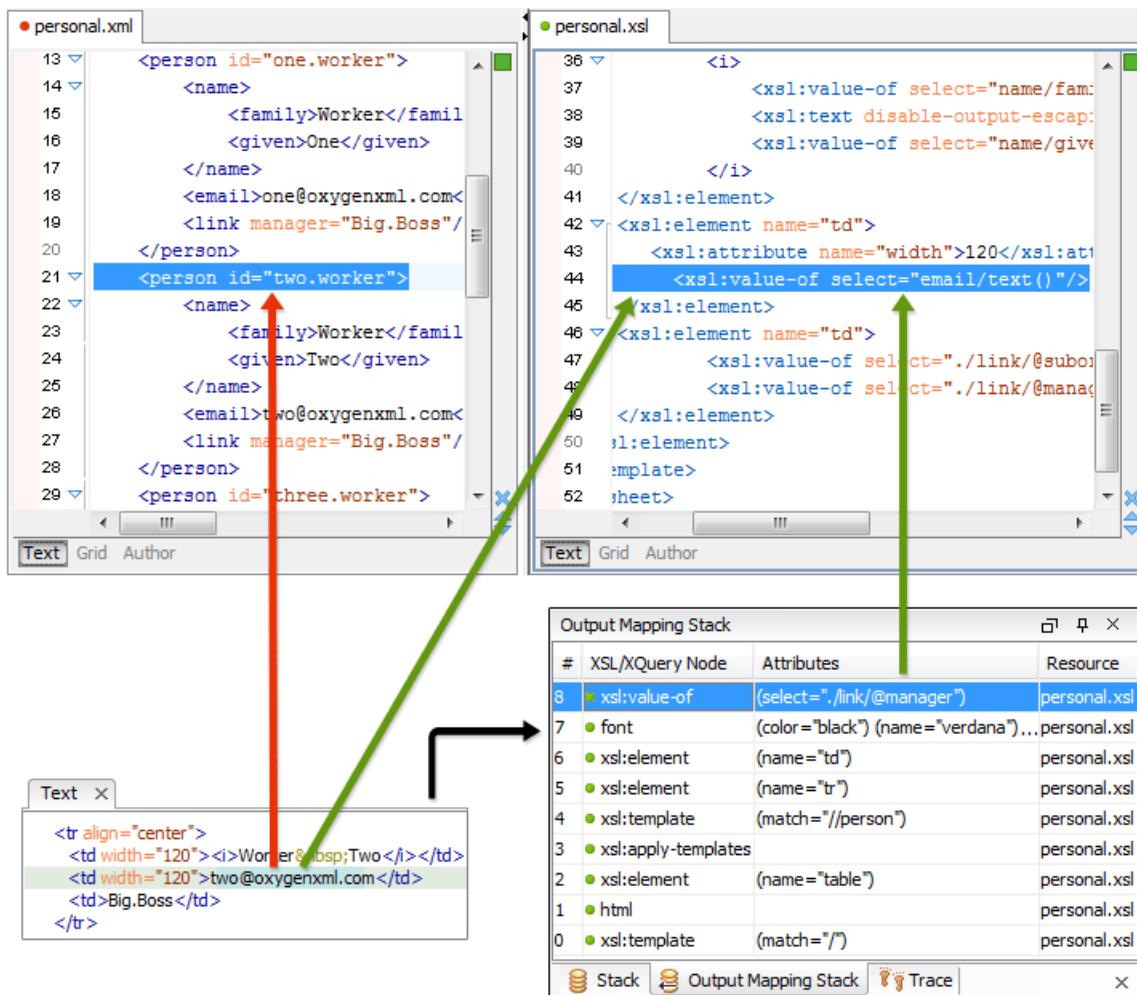
- Switch to the XSLT Debugger perspective or the XQuery Debugger perspective with one of the actions (here explained for XSLT):
  - Go to menu **Window > Open perspective > XSLT Debugger** or the toolbar button  **XSLT Debugger**
  - Go to menu **Document > XML Document > Debug scenario** or the toolbar button  **Debug scenario**. This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.

2. Select the source XML document in the XML source selector of *the Control toolbar*. In case of XQuery debugging without an implicit source choose the NONE value.
3. Select the XSLT / XQuery document in the XSLT / XQuery selector of *the Control toolbar*.
4. Select the XSLT / XQuery engine in the XSLT / XQuery engine selector of *the Control toolbar*.
5. Set XSLT / XQuery parameters from the button available on *the Control toolbar*.
6. Apply the XSLT stylesheet or XQuery transformation using the button **Run to end** available on *the Control toolbar*.
7. Inspect the mapping by clicking a section of the output from the **Text** view tab or from the **XHTML** view tab of the *Output document view*.

**Output Mapping Stack**

#	XSL/XQuery Node	Attributes	Resource
8	xsl:value-of	(select=".//link/@manager")	personal.xsl
7	font	(color="black") (name="verdana")...	personal.xsl
6	xsl:element	(name="td")	personal.xsl
5	xsl:element	(name="tr")	personal.xsl
4	xsl:template	(match="//person")	personal.xsl
3	xsl:apply-templates		personal.xsl
2	xsl:element	(name="table")	personal.xsl
1	html		personal.xsl
0	xsl:template	(match="/")	personal.xsl

Figure 317: XHTML Output to Source Mapping



**Figure 318: Text Output to Source Mapping**

This action will highlight the XSLT / XQuery element and the XML source context. This XSLT template/XQuery element that is highlighted in the XSLT/XQuery editor represents only the top of the stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the clicked output section. In case of complex transformations inspecting the whole stack of XSLT templates/XQuery elements speeds up the debugging process. This stack is available in [the Output Mapping Stack view](#).

## Debugging Java Extensions

The XSLT/XQuery debugger does not step into Java classes that are configured as XSLT/XQuery extensions of the transformation. To step into Java classes, inspect variable values and set breakpoints in Java methods, set up a Java debug configuration in an IDE like the Eclipse SDK as described in the following steps:

1. Create a debug configuration.
  - a) Set at least 256 MB as heap memory for the Java virtual machine (recommended 512 MB) by setting the `-Xmx` parameter in the debug configuration, for example “`-Xmx512m`”.
  - b) Make sure the `[OXYGEN_DIR]/lib/oxygen.jar` file and your Java extension classes are on the Java classpath.

The Java extension classes should be the same classes that were [set as an extension](#) of the XSLT/XQuery transformation in the debugging perspective.

- c) Set the class `ro.sync.exml.Oxygen` as the main Java class of the configuration.

The main Java class `ro.sync.exml.Oxygen` is located in the `oxygen.jar` file.

2. Start the debug configuration.

Now you can set breakpoints and inspect Java variables as in any Java debugging process executed in the selected IDE (Eclipse SDK, and so on.).

## Supported Processors for XSLT / XQuery Debugging

---

The following built-in XSLT processors are integrated in the debugger and can be selected in the [Control Toolbar](#):

- Saxon 9.6.0.5 HE (Home Edition) - a limited version of the Saxon 9 processor, capable of running XSLT 1.0, XSLT 2.0 / 3.0 basic and XQuery 1.0 transformations, available in both the XSLT debugger and the XQuery one,
- Saxon 9.6.0.5 PE (Professional Edition) - capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery one,
- Saxon 9.6.0.5 EE (Enterprise Edition) - a schema aware processor, capable of running XSLT 1.0 transformations, XSLT 2.0 / 3.0 basic ones, XSLT 2.0 / 3.0 schema aware ones and XQuery 1.0 / 3.0 ones, available in both the XSLT debugger and the XQuery debugger,
- Saxon 6.5.5 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger,
- Xalan 2.7.1 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger.



---

# Chapter

# 14

---

## Performance Profiling of XSLT Stylesheets and XQuery Documents

---

**Topics:**

- [\*Overview\*](#)
- [\*Viewing Profiling Information\*](#)
- [\*Working with XSLT/XQuery Profiler\*](#)

This chapter explains the user interface and how to use the profiler for finding performance problems in XSLT transformations and XQuery ones.

## Overview

---

Whether you are trying to identify a performance issue that is causing your production XSLT/XQuery transformation to not meet customer expectations or you are trying to proactively identify issues prior to deploying your XSLT/XQuery transformation, using the XSLT/XQuery profiler feature is essential to helping you save time and ultimately ensure a better performing, more scalable XSLT/XQuery transformation.

The XSLT/XQuery profiling feature can use any available XSLT/XQuery processors that could be used for debugging and it is available from the debugging perspective.

Enabling and disabling the profiler is controlled by the  **Profiler button** from the *debugger control toolbar*. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you should set it before starting the transformation.

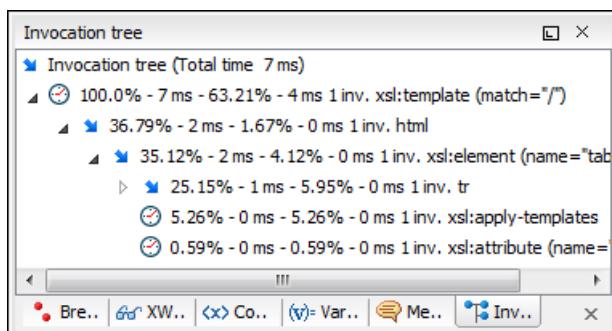
## Viewing Profiling Information

---

This section explains the views that display the profiling data collected by the profiles during the transformation.

### Invocation Tree View

This view shows a top-down call tree representing how XSLT instructions or XQuery expressions are processed.



**Figure 319: Invocation tree view**

The entries in the invocation tree have different meanings which are indicated by the displayed icons:

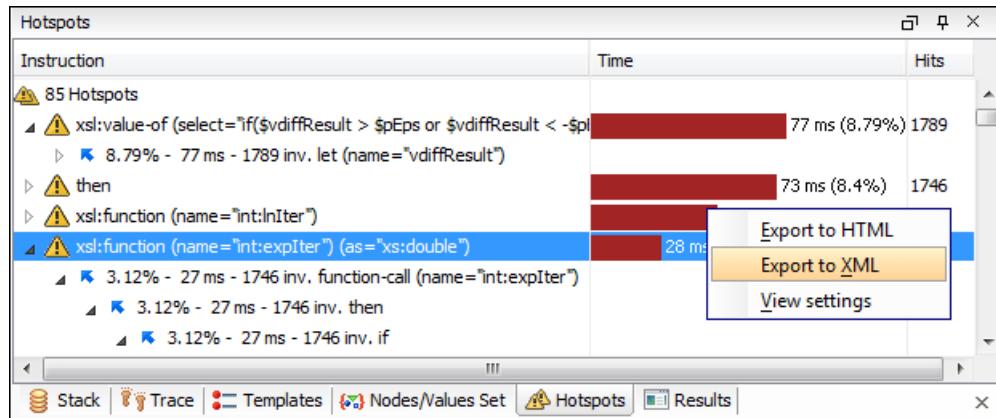
-  - Points to a call whose inherent time is insignificant compared to its total time.
-  - Points to a call whose inherent time is significant compared to its total time (greater than 1/3rd of its total time).

Every entry in the invocation tree has textual information attached which depends on the *XSLT/XQuery profiler settings*:

- A percentage number of total time which is calculated with respect to either the root of the tree or the calling instruction.
- A total time measurement in milliseconds or microseconds. This is the total execution time that includes calls into other instructions.
- A percentage number of inherent time which is calculated with respect to either the root of the tree or the calling instruction.
- An inherent time measurement in milliseconds or microseconds. This is the inherent execution time of the instruction.
- An invocation count which shows how often the instruction has been invoked on this call-path.
- An instruction name which contains also the attributes description.

## Hotspots View

This view shows a list of all instruction calls which lie above the threshold defined in the [XSLT/XQuery profiler settings](#).



**Figure 320: Hotspots View**

By opening a hotspot instruction entry, the tree of back-traces leading to that instruction call are calculated and shown.

Every hotspot is described by the values from the following columns:

- The instruction name.
- The inherent time in milliseconds or microseconds of how much time has been spent in the hotspot together with a bar whose length is proportional to this value. All calls into this instruction are summed up regardless of the particular call sequence.
- The invocation count of the hotspot.

If you click on the handle on the left side of a hotspot, a tree of back-traces will be shown.

Every entry in the backtrace tree has textual information attached to it which depends on the [XSLT/XQuery profiler settings](#):

- A percentage number which is calculated with respect either to the total time or the called instruction.
- A time measured in milliseconds or microseconds of how much time has been contributed to the parent hotspot on this call-path.
- An invocation count which shows how often the hotspot has been invoked on this call-path.



**Note:** This is not the number of invocations of this instruction.

- An instruction name which contains also its attributes.

## Working with XSLT/XQuery Profiler

Profiling activity is linked with debugging activity, so the first step in order to profile is to switch to debugging perspective and follow the corresponding procedure for debugging (see [Working with XSLT Debugger](#)).

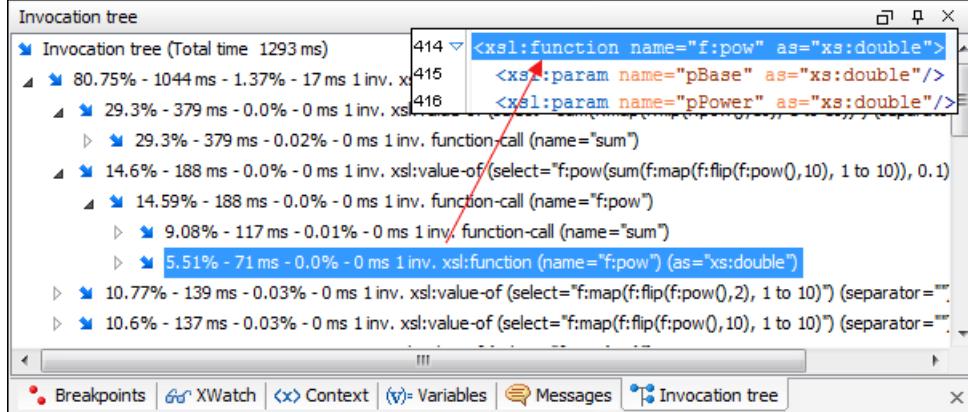
Immediately after turning the profiler on two new information views are added to the current debugger *information views*:

- [Invocation tree view](#) on left side
- [Hotspots view](#) on right side

Profiling data is available only after the transformation ends successfully.

Looking to the right side (***Hotspots view***), you can immediately spot the time the processor spent in each instruction. As an instruction usually calls other instructions the used time of the called instruction is extracted from the duration time of the caller (the hotspot only presents the inherent time of the instruction).

Looking to the left side (***Invocation tree view***), you can examine how style instructions are processed. This result view is also named call-tree, as it represents the order of style processing. The profiling result shows the duration time for each of the style-instruction including the time needed for its called children.



**Figure 321: Source backmapping**

In any of the above views you can use the backmapping feature in order to find the XSLT stylesheet or XQuery expression definition. Clicking on the selected item cause Oxygen XML Editor to highlight the XSLT stylesheet or XQuery expression source line where the instruction is defined.

When navigating through the trees by opening instruction calls, Oxygen XML Editor automatically expands instructions which are only called by one other instruction themselves.

The profiling data can be saved into XML and HTML format. On any view you should right click , use the pop-up menu and select the corresponding choice. Basically saving HTML means saving XML and applying an XSLT stylesheet to render the report as XML. These stylesheets are included in the Oxygen XML Editor distribution (see the subfolder [OXYGEN\_DIR]/frameworks/profiler/) so you can make your own report based on the profiling raw data.

If you like to change the **XSLT/XQuery profiler settings** you should right click on view, use the pop-up menu and choose the corresponding **View settings** entry.



**Caution:** Profiling exhaustive transformation may run into an OutOfMemory error due to the large amount of information being collected. If this is the case you can close unused projects when running the profiling or use high values for Java VM options –Xms and –Xmx. If this does not help you can shorten your source XML file and try again.

To watch our video demonstration about the XSLT/XQuery Profiler, go to  
[http://oxygenvml.com/demo/XSLT\\_Profiling.html](http://oxygenvml.com/demo/XSLT_Profiling.html).

---

# Chapter

# 15

---

## Working with Archives

---

### Topics:

- [\*Browsing and Modifying Archive Structure\*](#)
- [\*Working with EPUB\*](#)
- [\*Editing Files From Archives\*](#)

Oxygen XML Editor offers the means to manipulate files directly from ZIP type archives. By manipulation one should understand opening and saving files directly in archives, browsing and modifying archive structures. The archive support is available for all ZIP-type archives, which includes:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

This means that you can modify, transform, validate files directly from OOXML or ODF packages. The structure and content of an EPUB book, OOXML file or ODF file *can be opened, edited and saved* as for any other ZIP archive.

You can transform, validate and perform many other operations on files directly from an archive. When selecting an URL for a specific operation like transformation or validation you can click the  **Browse for archived file** button to navigate and choose the file from a certain archive.

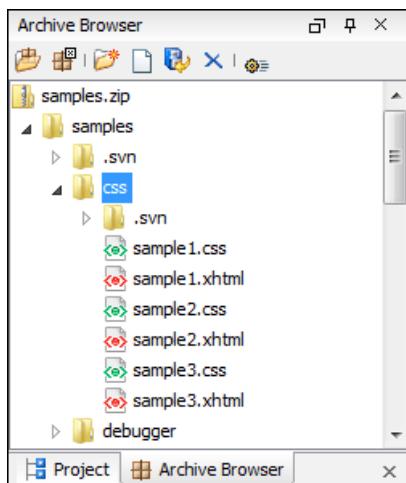
## Browsing and Modifying Archive Structure

You can open an archive in the **Archives Browser** view doing one of the following:

- Open an archive from the [Project view](#).
- Choose an archive in the Oxygen XML Editor file chooser dialog box.
- Drag an archive from the file explorer and drop it in the **Archives Browser** view.

When displaying an archive, the **Archive Browser** view locks the archive file. It is then automatically unlocked when the **Archive Browser** view is closed.

 **Important:** If a file is not recognized by Oxygen XML Editor as a supported archive type, you can add it from the [Archive preferences page](#).



**Figure 322: Browsing an Archive**

The following operations are available on the **Archive Browser** toolbar:



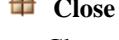
### Reopen

You can use this drop-down to reopen recently edited archives. Apart from the history of the recently edited archives, the drop-down also contains the **Clear history** and  **Open Archive** actions.



### Open Archive menu

Provides access to the  **Open Archive...** action which opens a new archive in the browser. If the extension is not known as an archive extension, you will be directed to the [Archive preferences page](#) to add a new extension. The sub-menu keeps a list of recently open archive files and a **Clear history** action which allows you to delete the list.



Closes the browsed archive and unlocks the archive file.



### New folder...

Creates a folder as child of the selected folder in the browsed archive.



Creates a file as child of the selected folder in the browsed archive.



Adds existing files as children of the selected folder in the browsed archive.



**Note:** You can also add files in the archive by dragging them from the file browser or **Project view** and dropping them in the **Archive Browser** view.

#### **Delete**

Deletes the selected resource in the browsed archive.

#### **Archive Options...**

Opens the *Archive preferences page*.

The following additional operations are available from the **Archive Browser** contextual menu:

#### **Open**

Opens a resource from the archive in the editor.

#### **Extract...**

Extracts a resource from the archive in a specified folder.

#### **New folder...**

Creates a folder as child of the selected folder in the browsed archive.

#### **New file...**

Creates a file as child of the selected folder in the browsed archive.

#### **Add files...**

Adds existing files as children of the selected folder in the browsed archive.

**Note:** On OS X, there is also available the **Add file...** action, which allows you to add one file at a time.

#### **Rename**

Renames a resource in the archive.

#### **Find/Replace in Files**

Allows you to search for and replace specific pieces of text inside the archive.

#### **Cut**

Cut the selected archive resource.

#### **Copy**

Copy the selected archive resource.

#### **Paste**

Paste a file or folder into the archive.

**Note:** You can add files in the archive by copying the files from the **Project view** and paste them into the **Archive view**.

#### **Delete**

Remove a file or folder from archive.

#### **Preview**

Previews an image contained in the archive See the *Image Preview* section for more details.

#### **Copy location**

Copies the URL location of the selected resource.

#### **Refresh**

Refreshes the selected resource.

#### **Properties**

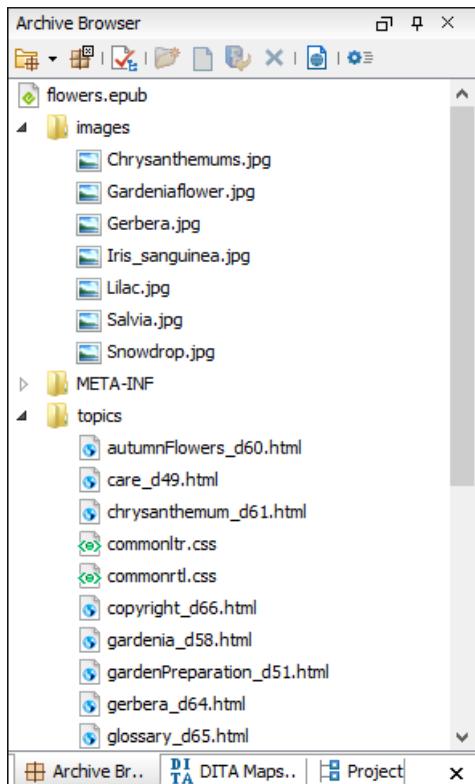
Views properties for the selected resource.

## Working with EPUB

**EPUB** is a free and open electronic book standard by the International Digital Publishing Forum (IDPF). It was designed for *reflowable content*, meaning that the text display can be optimized for the particular display device used by the reader of the EPUB-formatted book. Oxygen XML Editor supports both EPUB 2.0 and EPUB 3.0.

EPUB files are opened in the **Archive Browser** view, exposing all their internal components:

- Document content (XHTML and image files).
- Packaging files.
- Container files.



**Figure 323: EPUB file displayed in the Archive Browser view**

Here you can edit, delete and add files that compose the EPUB structure. To check that the EPUB file you are editing is valid, invoke the **Validate and Check for Completeness** action. Oxygen XML Editor uses the open-source EpubCheck validator to perform the validation. This validator detects many types of errors, including OCF container structure, OPF and OPS mark-up, as well as internal reference consistency. All errors found during validation are displayed in a separate tab in the **Errors** view.

- Note:** Invoke the **Open in System Application** action to see how the EPUB is rendered in your system default EPUB reader application.
- Note:** All changes made to the structure of an EPUB, or to the contents of the files inside an EPUB are immediately saved.

To watch our video demonstration about the EPUB support in Oxygen XML Editor, go to <http://oxygenvxml.com/demo/Epub.html>.

## Create an EPUB

To begin writing an EPUB file from scratch, do the following:

1. Go to **File > New**, press **Ctrl N (Command N on OS X)** on your keyboard, or click  **New** on the main toolbar.
2. Choose **EPUB Book** template. Click **Create**. Choose the name and location of the file. Click **Save**. A skeleton EPUB file is saved on disk and open in the **Archive Browser** view.
3. Use the **Archive Browser** view specific actions to edit, add and remove resources from the archive.
4. Use the  **Validate and Check for Completeness** action to verify the integrity of the EPUB archive.

## Publish to EPUB

Oxygen XML Editor comes with built-in support for publishing DocBook and DITA XML documents directly to EPUB.

1. Open the **Configure Transformation Scenario(s)** dialog box and select a predefined transformation scenario. To publish from DITA, select the **DITA Map EPUB** transformation scenario. To publish from DocBook select the **DocBook EPUB** transformation scenario.
2. Click **Apply associated** to run the transformation scenario.

## Editing Files From Archives

---

You can open and edit files directly from an archive using the **Archive Browser** view. When saving the file back to archive, you are prompted to choose if you want the application to make a backup copy of the archive before saving the new content. If you choose **Never ask me again**, you will not be asked again to make backup copies. You can re-enable the pop-up message from the [Messages preferences page](#).



**Note:** All changes made to the structure of an archive, or to the contents of the files inside an archive are immediately saved.



---

# Chapter

# 16

---

## Working with Databases

---

**Topics:**

- *Relational Database Support*
- *Native XML Database (NXD) Support*
- *XQuery and Databases*
- *WebDAV Connection*
- *BaseX Support*

XML is a storage and interchange format for structured data and is supported by all major database systems. Oxygen XML Editor offers the means for managing the interaction with some of the most commonly used databases (both relational and Native XML Databases). Through this interaction, Oxygen XML Editor helps users to understand browsing, querying, SQL execution support, content editing, importing from databases, and generating XML Schema from database structure.

## Relational Database Support

---

Relational databases use a relational model and are based on tables linked by a common key. Oxygen XML Editor offers support for the following relational databases: IBM DB2, MySQL, Microsoft SQL Server, and Oracle 11g.

The following actions are allowed:

- Browsing the tables of these types of databases in the **Data Source Explorer** view
- Executing SQL queries against them
- Calling stored procedures with input and output parameters

Oxygen XML Editor offers generic support (table browsing and execution of SQL queries) for any JDBC-compliant database (for example, *MariaDB*).

To watch our video demonstration about the integration between the relational databases and Oxygen XML Editor, go to [http://www.oxygenxml.com/demo/Author\\_Database\\_Integration.html](http://www.oxygenxml.com/demo/Author_Database_Integration.html).

## Configuring Database Data Sources

This section describes the procedures for configuring the data sources for relational databases:

- [IBM DB2](#)
- [Microsoft SQL Server](#)
- [Generic JDBC](#)
- [MySQL](#)
- [Oracle 11g](#)
- [PostgreSQL 8.3](#)

## Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

- [IBM DB2](#)
- [Microsoft SQL Server](#)
- [JDBC-ODBC](#)
- [MySQL](#)
- [Generic ODBC](#)
- [Oracle 11g](#)
- [PostgreSQL 8.3](#)

## How to Configure Support For Relational Databases

This section contains procedures about configuring the support for various relational databases.

### How to Configure IBM DB2 Support

To configure the support for the IBM DB2 database follow this procedure:

1. Go to the [IBM website](#) and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill out the download form and download the zip file. Unzip the zip file and use the `db2jcc.jar` and `db2jcc_license_cu.jar` files in Oxygen XML Editor for [configuring a DB2 data source](#).
2. [Configure a IBM DB2 Data Source driver](#).
3. [Configure a IBM DB2 Server Connection](#).
4. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

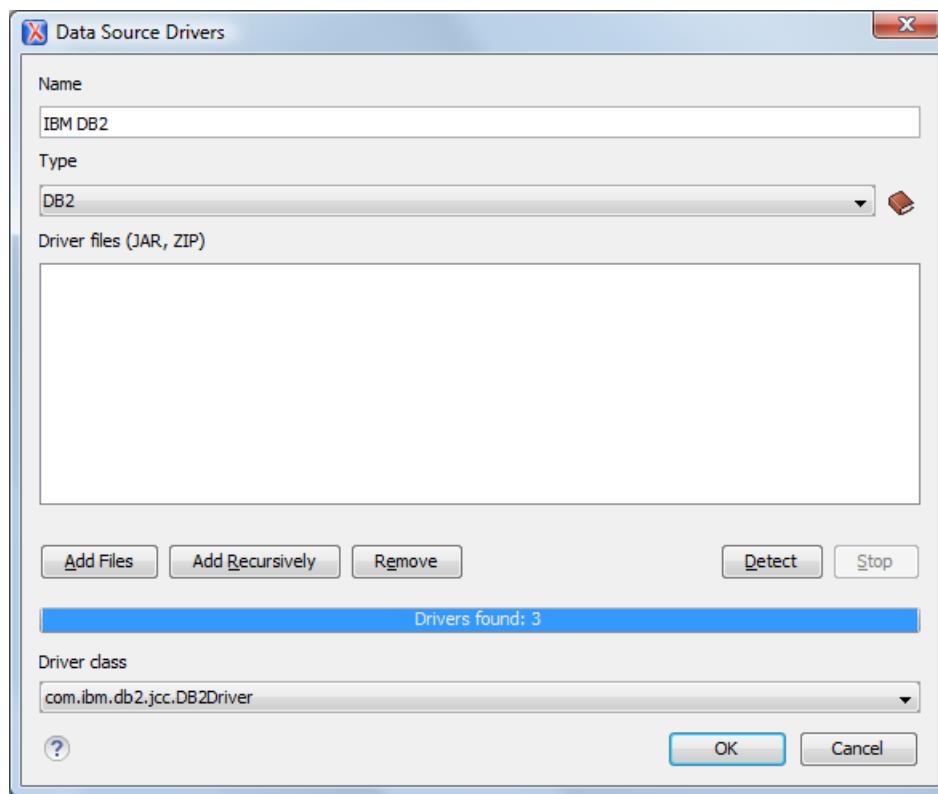
### How to Configure an IBM DB2 Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an IBM DB2 server are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the  New button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.



**Figure 324: Data Source Drivers Configuration Dialog Box**

3. Enter a unique name for the data source.
4. Select **DB2** in the driver **Type** drop-down list.
5. Add the driver files for IBM DB2 using the **Add Files** button.

The IBM DB2 driver files are:

- db2jcc.jar
- db2jcc\_license\_cisuz.jar
- db2jcc\_license\_cu.jar

The **Driver files** section lists [download links for database drivers](#) that are necessary for accessing IBM DB2 databases in Oxygen XML Editor.

6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to <http://www.oxygenxml.com/demo/DB2.html>.

### How to Configure an IBM DB2 Connection

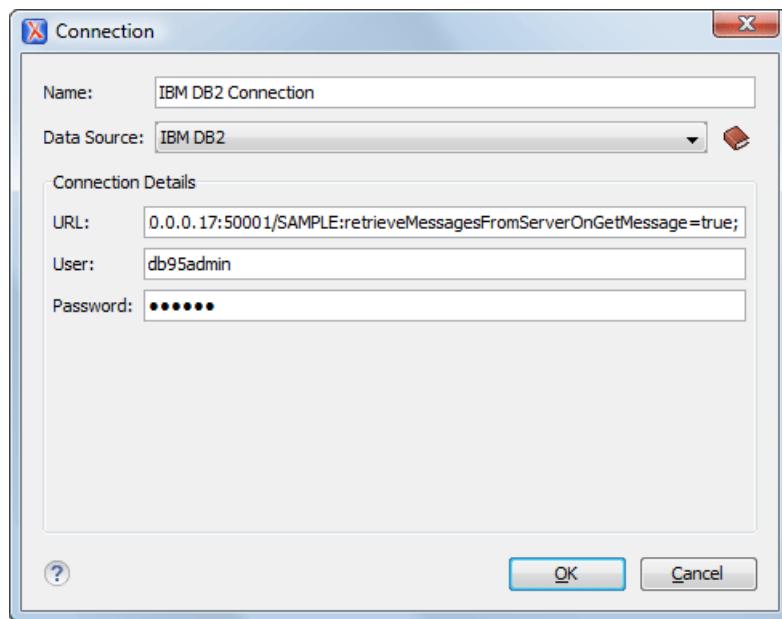
The support to create an IBM DB2 connection is available in the Enterprise edition only.

To configure a connection to an IBM DB2 server, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.

2. In the **Connections** panel, click the  **New** button.

The dialog box for configuring a database connection is displayed.



**Figure 325: The Connection Configuration Dialog Box**

3. Enter a unique name for the connection.
4. Select an *IBM DB2* data source in the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Enter the URL to the installed IBM DB2 engine.
  - b) Enter the user name to access the IBM DB2 engine.
  - c) Enter the password to access the IBM DB2 engine.
6. Click the **OK** button to finish the configuration of the database connection.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to <http://www.oxygenxml.com/demo/DB2.html>.

## How to Configure Microsoft SQL Server Support

To configure the support for Microsoft SQL Server database follow this procedure:

1. Download the appropriate MS SQL JDBC driver from the Microsoft website. For SQL Server 2008 R2 and older go to <http://www.microsoft.com/en-us/download/details.aspx?id=21599>. For SQL Server 2012 and 2014 go to <http://www.microsoft.com/en-us/download/details.aspx?id=11774>.
2. *Configure a MS SQL Server Data Source driver*.
3. *Configure a MS SQL Server Connection*.
4. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

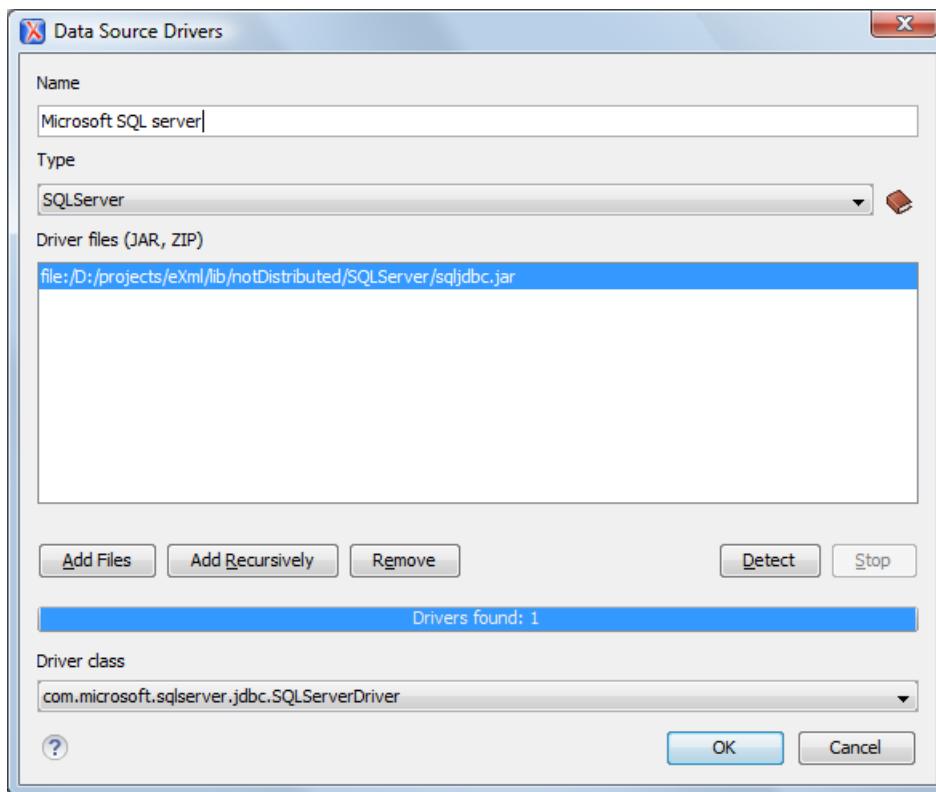
## How to Configure a Microsoft SQL Server Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to a Microsoft SQL server are as follows:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. Click the  **New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.



**Figure 326: Data Source Drivers Configuration Dialog Box**

3. Enter a unique name for the data source.
4. Select *SQLServer* in the driver **Type** drop-down list.
5. Add the Microsoft SQL Server driver file using the **Add Files** button.

The SQL Server driver file is called `sqljdbc.jar`. In the **Driver files** section lists [download links for database drivers](#) that are necessary for accessing Microsoft SQL Server databases in Oxygen XML Editor.

6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

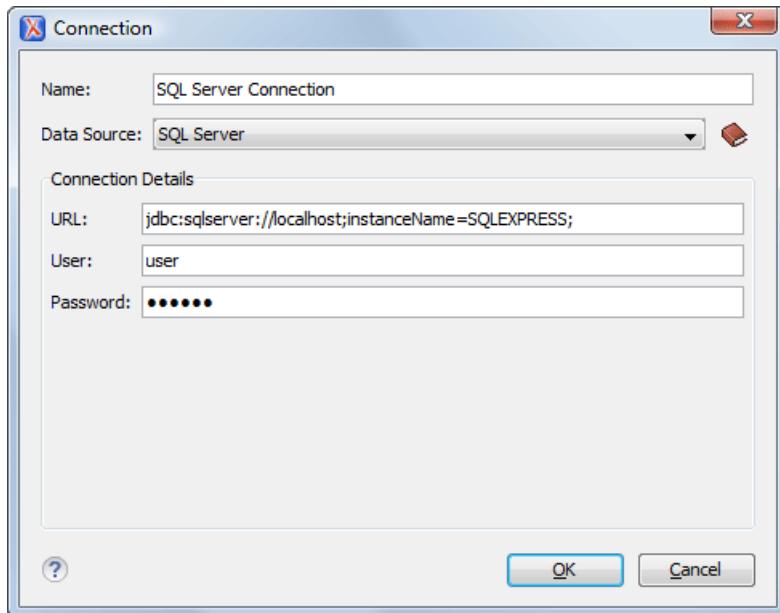
### How to Configure a Microsoft SQL Server Connection

The support to configure a Microsoft SQL Server connection is available in the Enterprise edition only.

To configure a connection to a Microsoft SQL Server, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

The dialog box for configuring a database connection is displayed.



**Figure 327: The Connection Configuration Dialog Box**

3. Enter a unique name for the connection.
4. Select the *SQL Server* data source in the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Enter the URL of the SQL Server server.

If you want to connect to the server using Windows integrated authentication, you must add `;integratedSecurity=true` to the end of the URL. The URL will look like this:

```
jdbc:sqlserver://localhost;instanceName=SQLEXPRESS;integratedSecurity=true;
```

 **Note:** For integrated authentication, leave the **User** and **Password** fields empty.

- b) Enter the user name for the connection to the SQL Server.
- c) Enter the password for the connection to the SQL Server.
6. Click the **OK** button to finish the configuration of the database connection.

### How to Configure Generic JDBC Support

To configure the support for a generic JDBC database follow this procedure:

1. [Configure a Generic JDBC Data Source driver](#).
2. [Configure a Generic JDBC Connection](#).
3. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

### How to Configure a Generic JDBC Data Source

Starting with version 17, Oxygen XML Editor comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Editor along with a Java VM version 7 or 6. The following procedure shows you how to configure a generic JDBC data source:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the  **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.

4. Select *Generic JDBC* in the driver **Type** drop-down list.
5. Add the driver file(s) using the **Add Files** button.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

### How to Configure a Generic JDBC Connection

To configure a connection to a generic JDBC database, follow these steps:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the  **New** button.
3. Enter a unique name for the connection.
4. Select the *Generic JDBC* data source in the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Enter the URL of the generic JDBC database, with the following format: `jdbc: <subprotocol>: <subname>`.
  - b) Enter the user name for the connection to the generic JDBC database.
  - c) Enter the password for the connection to the generic JDBC database.
6. Click the **OK** button to finish the configuration of the database connection.

### How to Configure MySQL Support

To configure the support for a MySQL database follow this procedure:

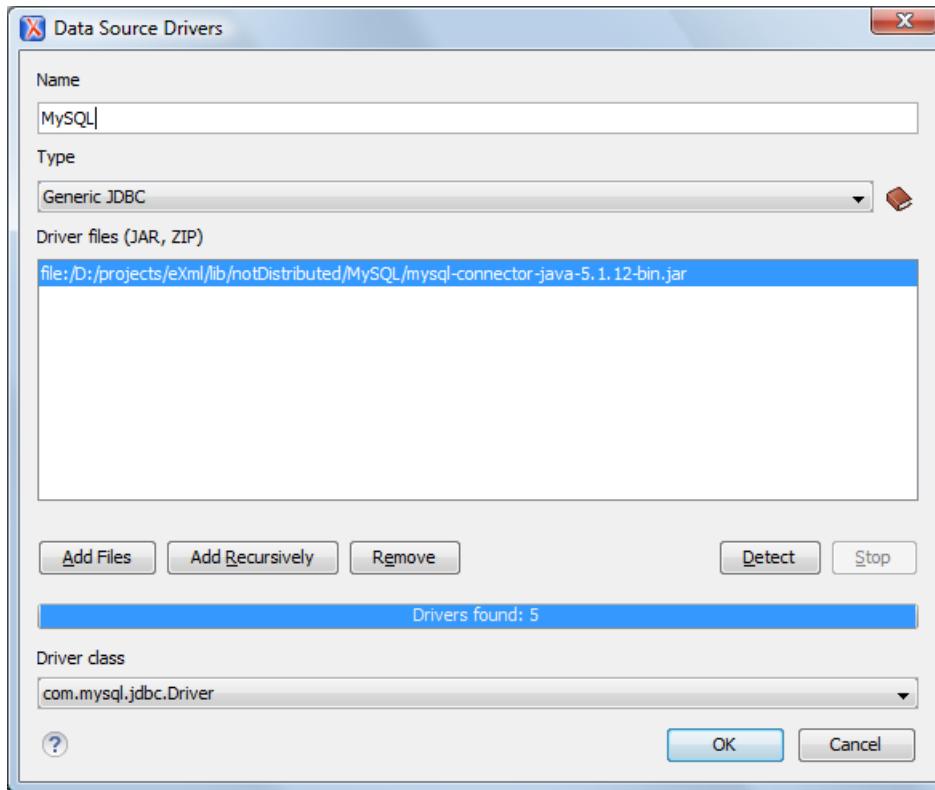
1. *Configure a MySQL Data Source driver*.
2. *Configure a MySQL Connection*.
3. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

### How to Configure a MySQL Data Source

To connect to a MySQL server, create a data source of a generic JDBC type, based on *the MySQL JDBC driver available on the MySQL website*. The following steps describe how you can configure such a data source:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. Click the  **New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.



**Figure 328: Data Source Drivers Configuration Dialog Box**

3. Enter a unique name for the data source.
4. Select *Generic JDBC* in the driver **Type** drop-down list.
5. Add the MySQL driver files using the **Add Files** button.

The driver file for the MySQL server is called `mysql-connector-java-5.1.12-bin.jar`. The **Driver files** section lists [download links for database drivers](#) that are necessary for accessing MySQL databases in Oxygen XML Editor.

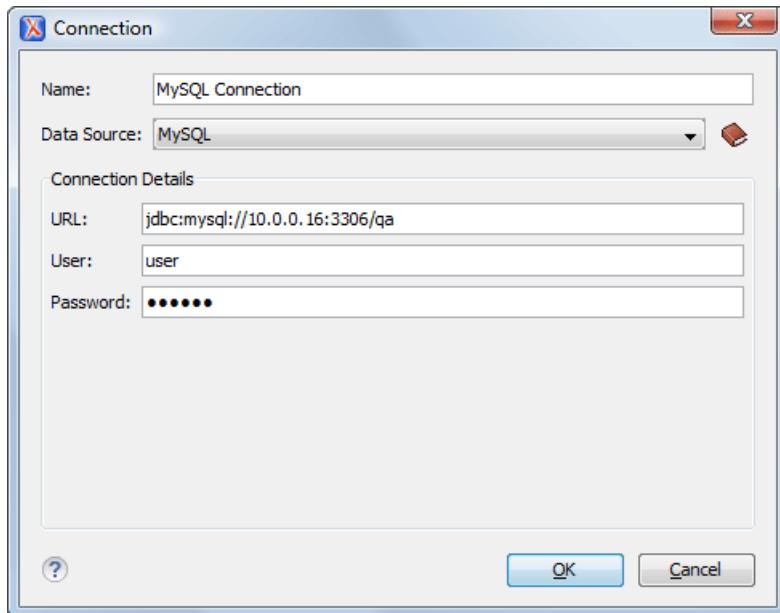
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

### How to Configure a MySQL Connection

To configure a connection to a MySQL server, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

The dialog box for configuring a database connection is displayed.



**Figure 329: The Connection Configuration Dialog Box**

3. Enter a unique name for the connection.
4. Select the **MySQL** data source in the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Enter the URL of the MySQL server.
  - b) Enter the user name for the connection to the MySQL server.
  - c) Enter the password for the connection to the MySQL server.
6. Click the **OK** button to finish the configuration of the database connection.

### How to Oracle 11g Support

To configure the support for a Oracle 11g database follow this procedure:

1. Go to the [Oracle website](#) and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
2. [Configure a Oracle 11g Data Source driver](#).
3. [Configure a Oracle 11g Connection](#).
4. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

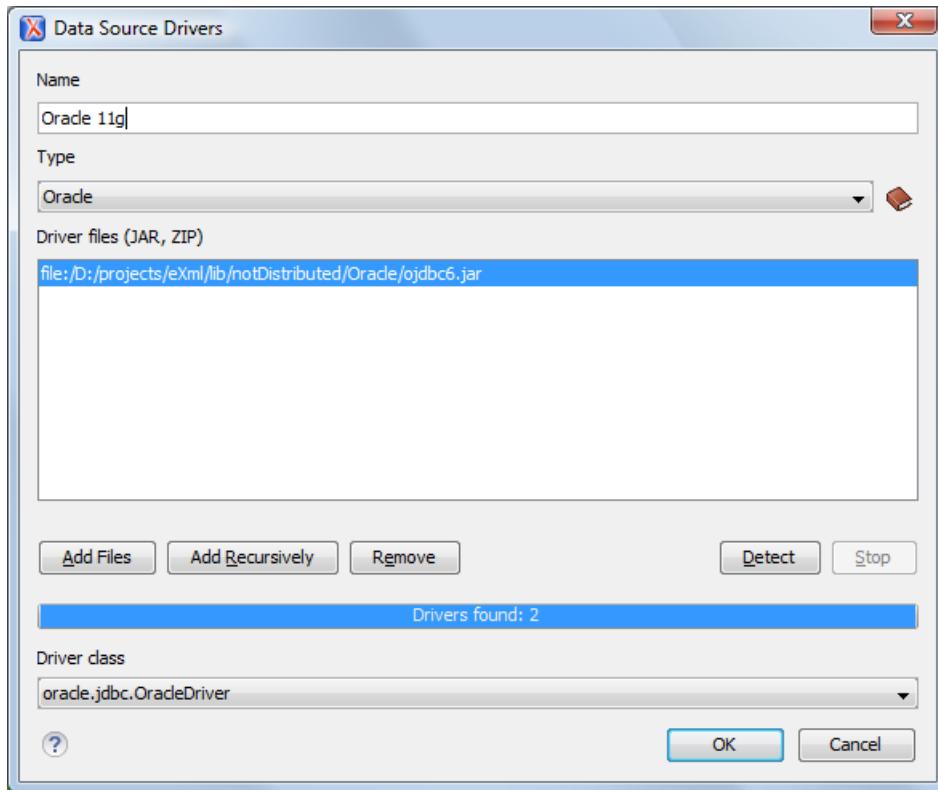
### How to Configure an Oracle 11g Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an Oracle 11g server are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.



**Figure 330: Data Source Drivers Configuration Dialog Box**

3. Enter a unique name for the data source.
4. Select *Oracle* in the driver **Type** drop-down list.
5. Add the Oracle driver file using the **Add Files** button.  
The Oracle driver file is called `ojdbc5.jar`. The **Driver files** section lists [download links for database drivers](#) that are necessary for accessing Oracle databases in Oxygen XML Editor.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

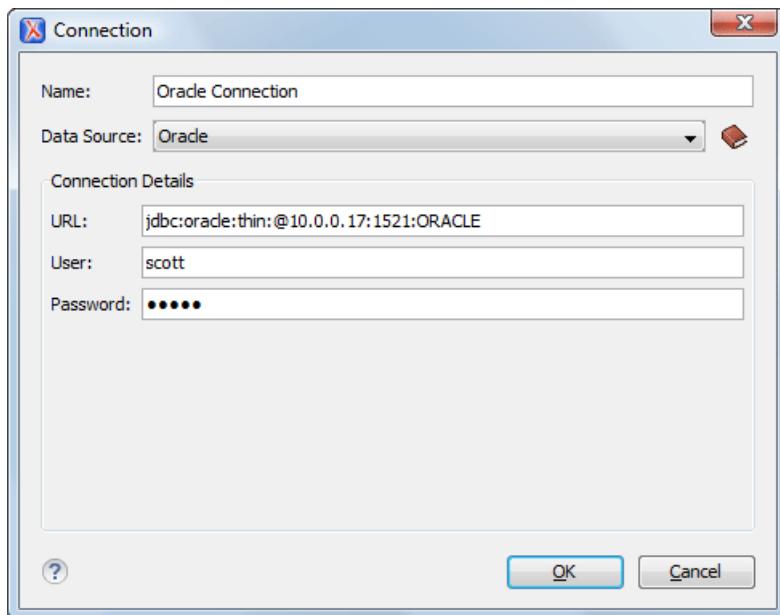
### How to Configure an Oracle 11g Connection

Available in the Enterprise edition only.

The steps for configuring a connection to an Oracle 11g server are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

The dialog box for configuring a database connection is displayed.



**Figure 331: The Connection Configuration Dialog Box**

3. Enter a unique name for the connection.
4. Select the *Oracle 11g* data source in the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Enter the URL of the Oracle server.
  - b) Enter the user name for the connection to the Oracle server.
  - c) Enter the password for the connection to the Oracle server.
6. Click the **OK** button to finish the configuration of the database connection.

### How to Configure PostgreSQL Support

To configure the support for a PostgreSQL database follow this procedure:

1. Go to the [PostgreSQL website](#) and download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar`.
2. [Configure a PostgreSQL Data Source driver](#).
3. [Configure a PostgreSQL Connection](#).
4. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

### How to Configure a PostgreSQL 8.3 Data Source

The steps for configuring a data source for connecting to a PostgreSQL server are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the New button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.

3. Enter a unique name for the data source.
4. Select *PostgreSQL* in the driver **Type** drop-down list.
5. Add the PostgreSQL driver file using the **Add Files** button.

The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`. The **Driver files** section lists [download links for database drivers](#) that are necessary for accessing PostgreSQL databases in Oxygen XML Editor.

6. Select the most appropriate **Driver class**.

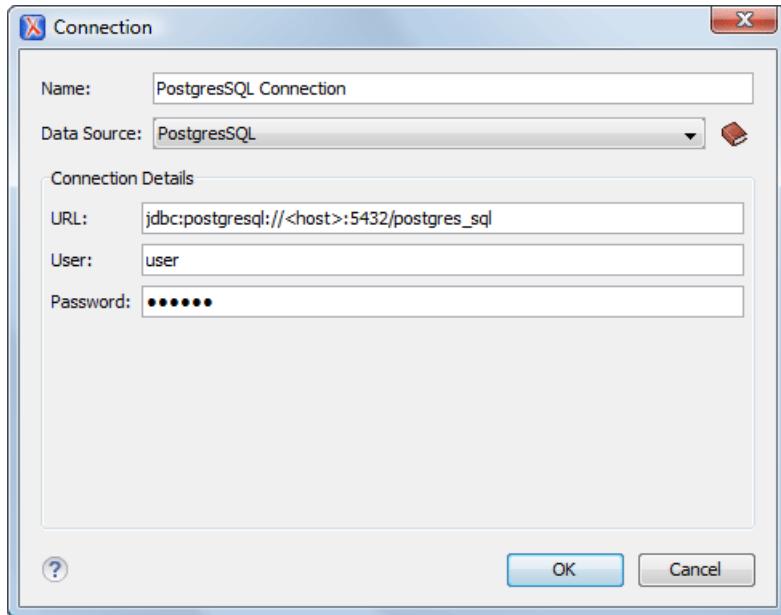
- Click the **OK** button to finish the data source configuration.

### How to Configure a PostgreSQL 8.3 Connection

The steps for configuring a connection to a PostgreSQL 8.3 server are as follows:

- [Open the Preferences dialog box](#) and go to **Data Sources**.
- In the **Connections** panel, click the **New** button.

The dialog box for configuring a database connection is displayed.



**Figure 332: The Connection Configuration Dialog Box**

- Enter a unique name for the connection.
- Select the *PostgreSQL 8.3* data source in the **Data Source** drop-down list.
- Enter the connection details.
  - Enter the URL of the PostgreSQL 8.3 server.
  - Enter the user name for the connection to the PostgreSQL 8.3 server.
  - Enter the password for the connection to the PostgreSQL 8.3 server.
- Click the **OK** button to finish the configuration of the database connection.

### How to Configure JDBC-ODBC Support

To configure the support for a JDBC-ODBC database follow this procedure:

- Configure a JDBC-ODBC *Data Source* driver.
- [Configure a JDBC-ODBC Connection](#).
- Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

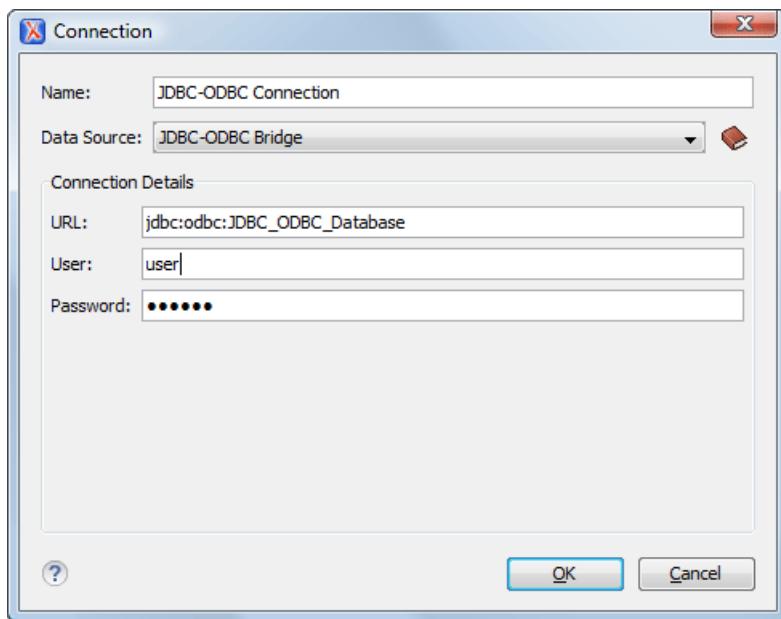
### How to Configure a JDBC-ODBC Connection

Starting with version 17, Oxygen XML Editor comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Editor along with a Java VM version 7 or 6. To configure a connection to an ODBC data source, follow these steps:

- [Open the Preferences dialog box](#) and go to **Data Sources**.

2. In the **Connections** panel, click the  **New** button.

The dialog box for configuring a database connection is displayed.



**Figure 333: The Connection Configuration Dialog Box**

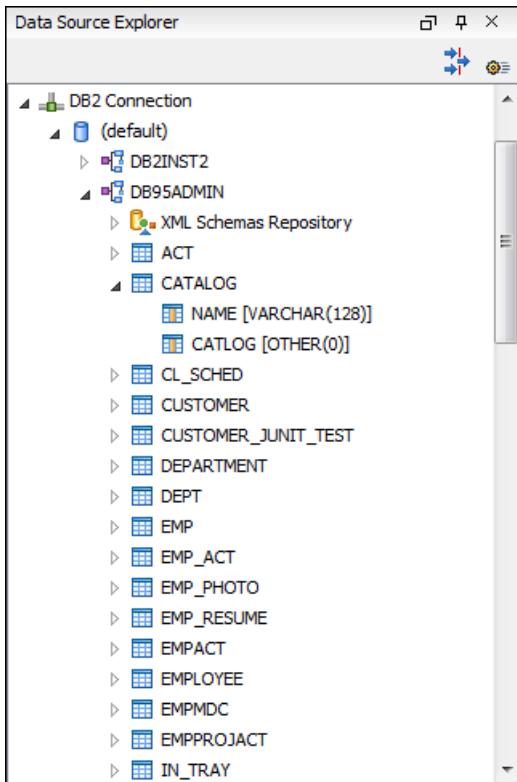
3. Enter a unique name for the connection.
4. Select *JDBC-ODBC Bridge* in the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Enter the URL of the ODBC source.
  - b) Enter the user name of the ODBC source.
  - c) Enter the password of the ODBC source.
6. Click the **OK** button to finish the configuration of the database connection.

## Resource Management

This section explains resource management actions for relational databases.

### Data Source Explorer View

The **Data Source Explorer** view displays your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded to the column level. Oxygen XML Editor supports multiple simultaneous database connections and the connection tree provides an easy method for browsing them.



**Figure 334: Data Source Explorer View**

The following objects are displayed in the **Data Source Explorer** view:

- **Connection**
- **Collection (Catalog)**
- **XML Schema Repository**
- **XML Schema Component**
- **Schema**
- **Table**
- **System Table**
- **Table Column**

A **Collection** (called *catalog* in some databases) is a hierarchical container for resources and sub-collections. There are two types of resources:

- **XML resource** - an XML document or document fragment, selected by a previously executed XPath query.
- **non-XML resource** -any resource that is not recognized as XML.

**Note:** For some connections you can add or move resources into a container by dragging them from:

- the **Project view**
- the default file system application (for example, Windows Explorer in Windows or Finder in Mac OS X)
- another database container

The following actions are available in the toolbar of this view:

**Filters**

Opens the **Data Sources / Table Filters** [Preferences page](#), allowing you to decide which table types are displayed in the **Data Source Explorer** view.

### **Configure Database Sources**

Opens the **Data Sources** [preferences page](#) where you can configure both data sources and connections.

### **Actions Available at Connection Level in Data Source Explorer View**

The contextual menu of a  **Connection** node in the tree from the **Data Source Explorer** view contains the following actions:

#### **Refresh**

Performs a refresh for the sub-tree of the selected node.

#### **Disconnect**

Closes the current database connection. If a table is already open, you are warned to close it before proceeding.

### **Configure Database Sources**

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

### **Actions Available at Catalog Level in Data Source Explorer View**

The contextual menu of a  **Catalog** node in the tree from the **Data Source Explorer** view contains the following actions:

#### **Refresh**

Performs a refresh for the sub-tree of the selected node.

### **Actions Available at Schema Level in Data Source Explorer View**

The contextual menu of a  **Schema** node in the tree from the **Data Source Explorer** view contains the following actions:

#### **Refresh**

Performs a refresh for the sub-tree of the selected node.

### **Actions Available at Table Level in Data Source Explorer View**

The contextual menu of a  **Table** node in the tree from the **Data Source Explorer** view contains the following actions:

#### **Refresh**

Performs a refresh for the sub-tree of the selected node.

#### **Edit**

Opens the selected table in the **Table Explorer** view.

#### **Export to XML**

Opens the **Export Criteria** dialog (a thorough description of this dialog can be found in the [Import from Database](#) chapter).

### **XML Schema Repository Level**

This section explains the actions available at the XML Schema Repository level.

#### *Oracle's XML Schema Repository Level*

The Oracle database supports XML schema repository (XSR) in the database catalogs. The contextual menu of a  **XML Schema Repository** node in the tree from the **Data Source Explorer** view contains the following actions:

#### **Refresh**

Performs a refresh for the sub-tree of the selected node.

#### **Register**

Opens a dialog for adding a new schema file in the XML repository. To add an XML Schema, enter the schema URI and location on your file system. *Local* scope means that the schema is visible only to the user who registers it. *Global* scope means that the schema is public.



**Note:** Registering a schema may involve dropping/creating types. Hence you need type-related privileges such as DROP TYPE, CREATE TYPE, and ALTER TYPE. You need privileges to delete and register the XML schemas involved in the registering process. You need all privileges on XMLType tables that conform to the registered schemas. For XMLType columns, the ALTER TABLE privilege is needed on corresponding tables. If there are schema-based XMLType tables or columns in other database schemas, you need privileges such as the following:

- CREATE ANY TABLE
- CREATE ANY INDEX
- SELECT ANY TABLE
- UPDATE ANY TABLE
- INSERT ANY TABLE
- DELETE ANY TABLE
- DROP ANY TABLE
- ALTER ANY TABLE
- DROP ANY INDEX

To avoid having to grant all these privileges to the schema owner, Oracle recommends that the registration be performed by a DBA if there are XML schema-based XMLType table or columns in other user database schemas.

#### *IBM DB2's XML Schema Repository Level*

The contextual menu of a **XML Schema Repository** node in the tree from the **Data Source Explorer** view contains the following actions:

##### **Refresh**

Performs a refresh for the sub-tree of the selected node.

##### **Register**

Opens a dialog box for adding a new schema file in the XML Schema repository. In this dialog box, the following fields can be set:

- **XML schema file** - Location on your file system.
- **XSR name** - Schema name.
- **Comment** - Short comment (optional).
- **Schema location** - Primary schema name (optional).

Decomposition means that parts of the XML documents are stored in relational tables. Which parts map to which tables and columns are specified in the schema annotations. Schema dependencies management is done by using the **Add** and **Remove** buttons.

The actions available at **Schema** level are as follows:

##### **Refresh**

Performs a refresh of the selected node (and its sub-tree).

##### **Unregister**

Removes the selected schema from the XML Schema Repository.

##### **View**

Opens the selected schema in Oxygen XML Editor.

## Microsoft SQL Server's XML Schema Repository Level

The contextual menu of a  **XML Schema Repository** node in the tree from the **Data Source Explorer** view contains the following actions:

### Refresh

Performs a refresh for the sub-tree of the selected node.

### Register

Opens a dialog for adding a new schema file in the DB XML repository. In this dialog you enter a collection name and the necessary schema files. XML Schema files management is done by using the **Add** and **Remove** buttons.

The actions available at  **Schema** level are as follows:

### Refresh

Performs a refresh of the selected node (and its sub-tree).

### Add

Adds a new schema to the XML Schema files.

### Unregister

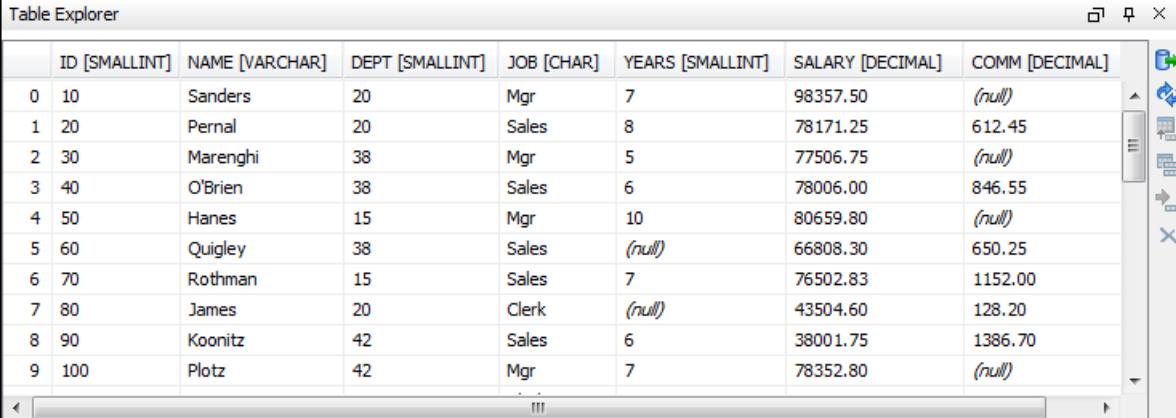
Removes the selected schema from the XML Schema Repository.

### View

Opens the selected schema in Oxygen XML Editor.

## Table Explorer View

Every table from the **Data Source Explorer** view can be displayed and edited in the **Table Explorer** view by pressing the **Edit** button from the contextual menu or by double-clicking one of its fields. To modify the content of a cell, double-click it and start typing. When editing is complete, Oxygen XML Editor attempts to update the database with the new cell content.



	ID [SMALLINT]	NAME [VARCHAR]	DEPT [SMALLINT]	JOB [CHAR]	YEARS [SMALLINT]	SALARY [DECIMAL]	COMM [DECIMAL]
0	10	Sanders	20	Mgr	7	98357.50	(null)
1	20	Pernal	20	Sales	8	78171.25	612.45
2	30	Marenghi	38	Mgr	5	77506.75	(null)
3	40	O'Brien	38	Sales	6	78006.00	846.55
4	50	Hanes	15	Mgr	10	80659.80	(null)
5	60	Quigley	38	Sales	(null)	66808.30	650.25
6	70	Rothman	15	Sales	7	76502.83	1152.00
7	80	James	20	Clerk	(null)	43504.60	128.20
8	90	Koonitz	42	Sales	6	38001.75	1386.70
9	100	Plotz	42	Mgr	7	78352.80	(null)

**Figure 335: The Table Explorer View**

You can sort the content of a table by one of its columns by clicking on its column header.

Note the following:

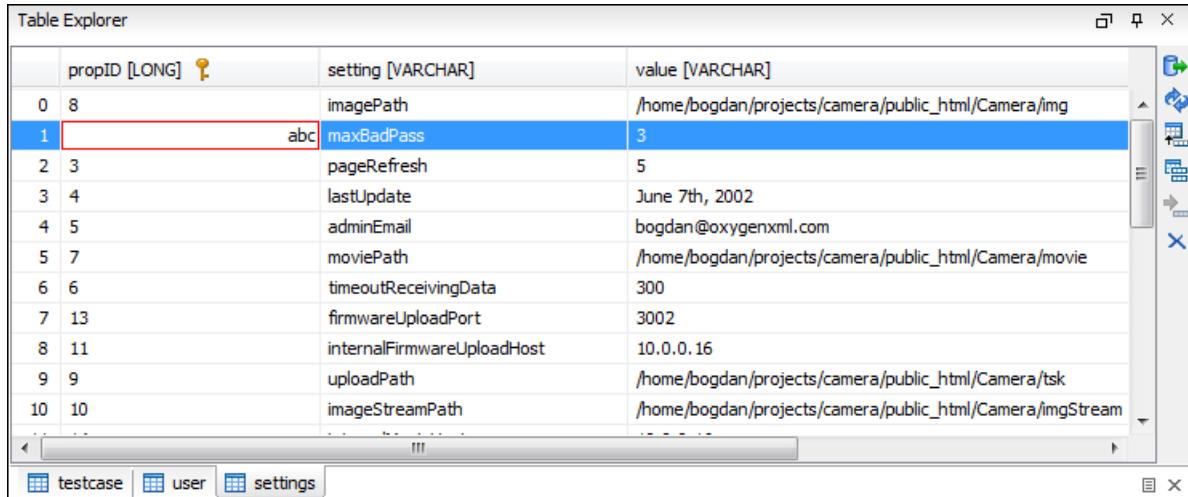
- The first column is an index (not part of the table structure)
- Every column header contains the field name and its data type
- The primary key columns are marked with this symbol: 
- Multiple tables are presented in a tabbed manner

For performance issues, you can set the maximum number of cells that are displayed in the **Table Explorer** view (using the **Limit the number of cells** field from the [Data Sources](#) Preferences page). If a table that has more cells than the

value set in the options is displayed in the **Table Explorer** view, a warning dialog informs you that the table is only partially shown.

You are notified if the value you have entered in a cell is not valid (and thus cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, the cell is marked by a red square and remains in an editing state until a correct value is inserted. For example, in the following figure `propID` contains `LONG` values. If a character or string is inserted, the cell will look like this:

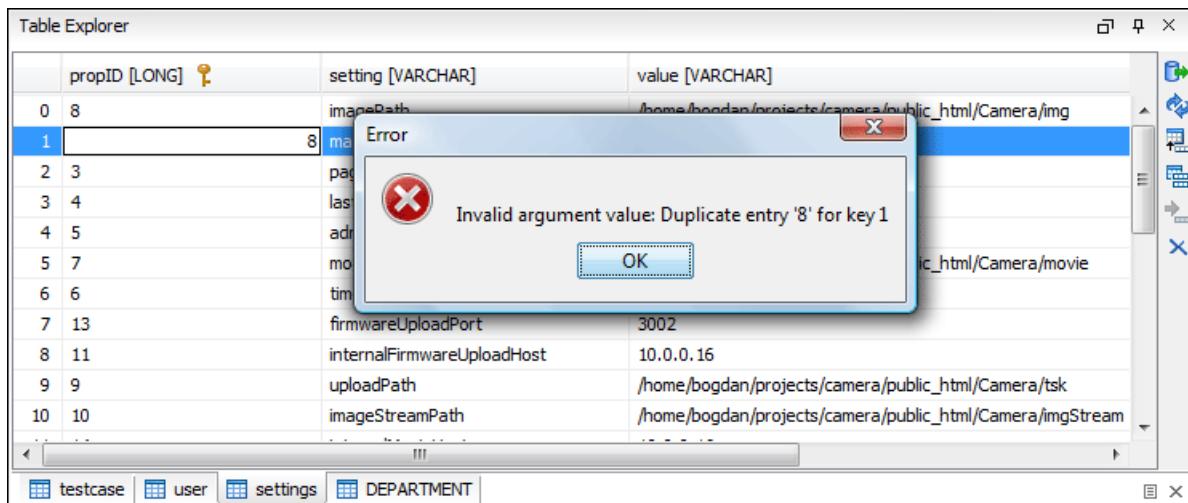


The screenshot shows the Table Explorer interface with a table named 'settings'. The columns are 'propID [LONG]', 'setting [VARCHAR]', and 'value [VARCHAR]'. Row 1 has a value 'abc' in the 'propID' column, which is highlighted with a red border. The other rows contain valid data such as 'imagePath' and 'value' paths.

	propID [LONG]	setting [VARCHAR]	value [VARCHAR]
0	8	imagePath	/home/bogdan/projects/camera/public_html/Camera/img
1	abc	maxBadPass	3
2	3	pageRefresh	5
3	4	lastUpdate	June 7th, 2002
4	5	adminEmail	bogdan@oxygentools.com
5	7	moviePath	/home/bogdan/projects/camera/public_html/Camera/movie
6	6	timeoutReceivingData	300
7	13	firmwareUploadPort	3002
8	11	internalFirmwareUploadHost	10.0.0.16
9	9	uploadPath	/home/bogdan/projects/camera/public_html/Camera/tsk
10	10	imageStreamPath	/home/bogdan/projects/camera/public_html/Camera/imgStream

**Figure 336: Cell Containing an Invalid Value**

- If the constraints of the database are not met (for instance, primary key constraints), an information dialog will appear, notifying you of the reason the database has not been updated. For example, in the table below, trying to set the second record in the primary key `propID` column to 8, results in a duplicate entry error since that value has already been used in the first record:



The screenshot shows the Table Explorer interface with a table named 'settings'. The columns are 'propID [LONG]', 'setting [VARCHAR]', and 'value [VARCHAR]'. Row 1 has a value '8' in the 'propID' column. A modal dialog box titled 'Error' is displayed, stating 'Invalid argument value: Duplicate entry '8' for key 1'. The 'OK' button is visible at the bottom of the dialog. The rest of the table data is visible in the background.

	propID [LONG]	setting [VARCHAR]	value [VARCHAR]
0	8	imagePath	/home/bogdan/projects/camera/public_html/Camera/img
1	8	maxBadPass	Error
2	3	pageRefresh	
3	4	lastUpdate	
4	5	adminEmail	
5	7	moviePath	
6	6	timeoutReceivingData	
7	13	firmwareUploadPort	3002
8	11	internalFirmwareUploadHost	10.0.0.16
9	9	uploadPath	/home/bogdan/projects/camera/public_html/Camera/tsk
10	10	imageStreamPath	/home/bogdan/projects/camera/public_html/Camera/imgStream

**Figure 337: Duplicate Entry for Primary Key**

Common edit actions (**Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo**) are available in the popup menu of an edited cell.

The contextual menu, available on every cell, also has the following actions:

#### Set NULL

Sets the content of the cell to `null`. This action is disabled for columns that cannot have a value of `null`.

 **Insert row**

Inserts an empty row in the table.

 **Duplicate row**

Makes a copy of the selected row and adds it in the **Table Explorer** view. Note that the new row will not be inserted in the database table until all conflicts are resolved.

 **Commit row**

Commits the selected row.

 **Delete row**

Deletes the selected row.

 **Copy**

Copies the content of the cell.

 **Paste**

Pastes copied content into the selected cell.

The **Table Explorer** toolbar also includes the following actions:

 **Export to XML**

Opens the **Export Criteria** dialog (a thorough description of this dialog can be found in the *Import from database* chapter).

 **Refresh**

Performs a refresh for the sub-tree of the selected node.

 **Insert row**

Inserts an empty row in the table.

 **Duplicate row**

Makes a copy of the selected row and adds it in the **Table Explorer** view. Note that the new row will not be inserted in the database table until all conflicts are resolved.

 **Commit row**

Commits the selected row.

 **Delete row**

Deletes the selected row.

## SQL Execution Support

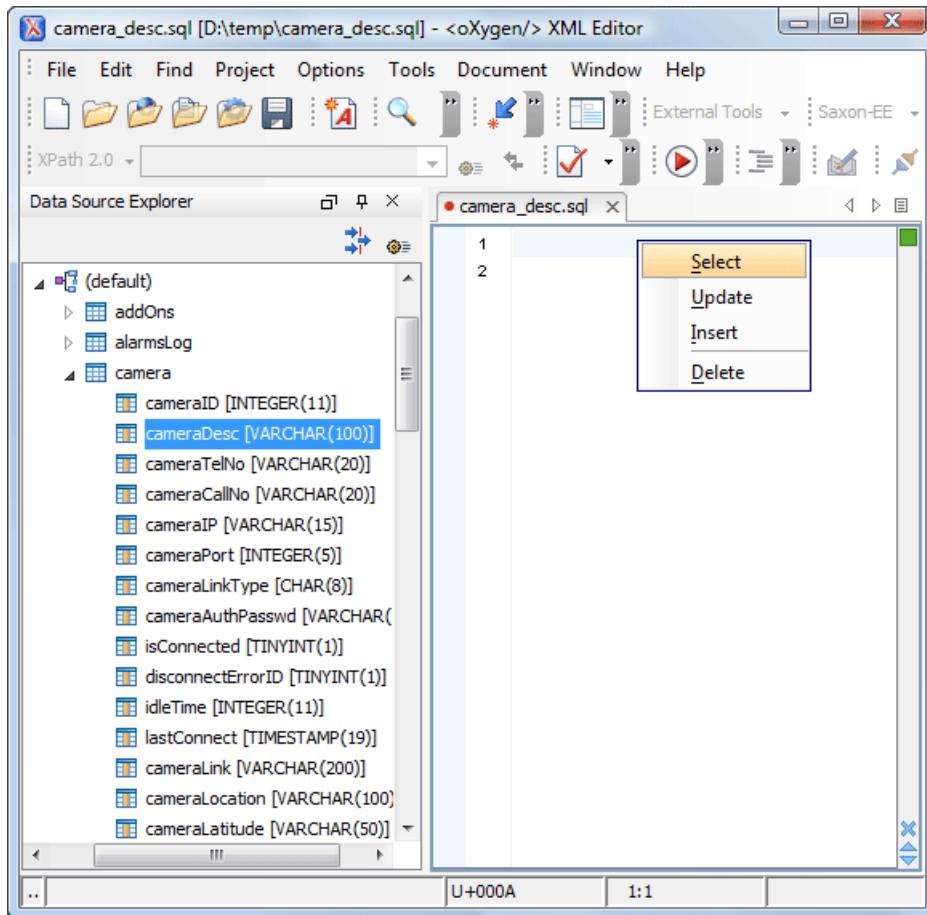
Oxygen XML Editor's support for writing SQL statements includes syntax highlighting, folding, and dragging and dropping from the **Data Source Explorer** view. It also includes transformation scenarios for executing the statements, and the results are displayed in the **Table Explorer** view.

### Drag and Drop from Data Source Explorer View

Drag and drop operations from the **Data Source Explorer** view to the SQL Editor allows you to create SQL statements quickly by inserting the names of tables and columns in the SQL statements.

1. Configure a database connection (*see the specific procedure for your database server*).
2. Browse to the table you will use in your statement.
3. Drag the table or a column of the table into the editor where a SQL file is open.

Drag and drop actions are available both on the table and on its fields. A pop-up menu is displayed in the SQL editor.



**Figure 338: SQL Statement Editing with Drag and Drop**

4. Select the type of statement from the pop-up menu.

Depending on your choice, dragging a table results in one of the following statements being inserted into the document:

- `SELECT `field1`, `field2`, .... FROM `catalog`.`table`` (for example: `SELECT `DEPT`, `DEPTNAME`, `LOCATION` FROM `camera`.`cameraDesc``)
- `UPDATE `catalog`.`table` SET `field1` = , `field2` = , ....` (for example: `UPDATE `camera`.`cameraDesc` SET `DEPT` = , `DEPTNAME` = , `LOCATION` = )`
- `INSERT INTO `catalog`.`table` (`field1`, `field2`, ....) VALUES (, , )` (for example: `INSERT INTO `camera`.`cameraDesc` (`DEPT`, `DEPTNAME`, `LOCATION`) VALUES (, , )`)
- `DELETE FROM `catalog`.`table`` (for example: `DELETE FROM `camera`.`cameraDesc``)

Depending on your choice, dragging a column results in one of the following statements being inserted into the document:

- `SELECT `field` FROM `catalog`.`table`` (for example: `SELECT `DEPT` FROM `camera`.`cameraDesc``)
- `UPDATE `catalog`.`table` SET `field` = (for example: UPDATE `camera`.`cameraDesc` SET `DEPT` = )`
- `INSERT INTO `catalog`.`table` (`field1`) VALUES ()` (for example: `INSERT INTO `camera`.`cameraDesc` (`DEPT`) VALUES ()`)
- `DELETE FROM `catalog`.`table`` (for example: `DELETE FROM `camera`.`cameraDesc` WHERE `DEPT` = )`

## SQL Validation

SQL validation support is offered for IBM DB2. Please note that if you choose a connection that does not support SQL validation, you will receive a warning when trying to validate. The SQL document is validated using the connection from the associated transformation scenario.

## Executing SQL Statements

The steps for executing an SQL statement on a relational database are as follows:

1. Configure a *transformation scenario* using the  **Configure Transformation Scenario(s)** action from the **Transformation** toolbar or the **Document > Transformation** menu.

A SQL transformation scenario needs a database connection. You can configure a connection using the  **Preferences** button from the SQL transformation dialog box.

The dialog box contains the list of existing scenarios that apply to SQL documents.

2. Set parameter values for SQL placeholders using the **Parameters** button from the SQL transformation dialog box. For example, in `SELECT * FROM `test`.`department` where DEPT = ? or DEPTNAME = ?` the two parameters can be configured for the place holders (?) in the transformation scenario.

When the SQL statement is executed, the first placeholder is replaced with the value set for the first parameter in the scenario, the second placeholder is replaced by the second parameter value, and so on.

 **Restriction:** When a stored procedure is called in an SQL statement executed on an SQL Server database, mixing in-line parameter values with values specified using the **Parameters** button of the scenario dialog box is not recommended. This is due to a limitation of the SQL Server driver for Java applications. An example of stored procedure that is not recommended: `call dbo.Test(22, ?)`.

3. Execute the SQL scenario by clicking the **OK** or **Apply associated** button.

The result of a SQL transformation is *displayed in a view* at the bottom of the Oxygen XML Editor window.

4. View more complex return values of the SQL transformation in a separate editor panel.

A more complex value returned by the SQL query (for example, an `XMLTYPE` or `CLOB` value) cannot be displayed entirely in the result table.

- a) Right-click on the cell containing the complex value.
- b) Select the action **Copy cell** from the contextual menu.  
The action copies the value in the clipboard.
- c) Paste the value into an appropriate editor.

For example, you can paste the value in an opened XQuery editor panel of Oxygen XML Editor.

## Native XML Database (NXD) Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. Oxygen XML Editor offers support for the following native XML databases:

- Berkeley DB XML
- eXist
- MarkLogic
- Documentum xDb (X-Hive/DB) 10
- Oracle XML DB

To watch our video demonstration about the integration between the XML native databases and Oxygen XML Editor, go to [http://www.oxygenxml.com/demo/Author\\_Database\\_XML\\_Native.html](http://www.oxygenxml.com/demo/Author_Database_XML_Native.html).

## Configuring Database Data Sources

This section describes the procedures for configuring the following native database data sources:

- [Berkeley DB XML](#)
- [eXist](#)
- [MarkLogic](#)
- [Documentum xDb \(X-Hive/DB\) 10](#)

## Configuring Database Connections

This section describes the procedures for configuring the connections for the following native databases:

- [Berkeley DB XML](#)
- [eXist](#)
- [MarkLogic](#)
- [Documentum xDb \(X-Hive/DB\) 10](#)

## How to Configure Support for Native XML Databases

This section contains procedures about configuring the support for various native XML databases.

### How to Configure Berkeley DB XML Support

Follow this procedure to configure the support for a Berkeley DB XML database:

1. [Configure a Berkeley DB XML Data Source driver.](#)
2. [Configure a Berkeley DB XML Connection.](#)
3. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

### How to Configure a Berkeley DB XML Data Source

Oxygen XML Editor supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a data source for a Berkeley DB XML database are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *Berkeley DBXML* from the driver **Type** drop-down list.
5. Click the **Add** button to add the Berkeley DB driver files.

The driver files for the Berkeley DB database are the following:

- db.jar (check for it in [DBXML\_DIR] / lib or [DBXML\_DIR] / jar)
- dbxml.jar (check for it in [DBXML\_DIR] / lib or [DBXML\_DIR] / jar)

Where [DBXML\_DIR] is the Berkeley DB XML database root directory. For example, in Windows it is:  
C:\Program Files\Oracle\Berkeley DB XML <version>.

6. Click the **OK** button to finish the data source configuration.

### How to Configure a Berkeley DB XML Connection

Oxygen XML Editor supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a connection to a Berkeley DB XML database are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down list.
5. Enter the connection details.

- a) Set the path to the Berkeley DB XML database directory in the **Environment home directory field**. Use a directory with write access. DO NOT use the installation directory where Berkeley DB XML is installed if you do not have write access to that directory.
- b) Select the **Verbosity** level: *DEBUG, INFO, WARNING, or ERROR*.
- c) Optionally, you can select the check-box **Join existing environment**. If checked, an attempt is made to join an existing environment in the specified home directory and all the original environment settings are preserved. If that fails, try reconfiguring the connection with this option unchecked.

6. Click the **OK** button to finish the connection configuration.

## How to Configure eXist Support

Follow this procedure to configure the support for an eXist database:

1. *Configure a eXist Data Source driver.*
2. *Configure a eXist Connection.*
3. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

## How to Configure an eXist Data Source

Oxygen XML Editor supports eXist database server versions up to and including version 2.2. The steps for configuring a data source for an eXist database are as follows:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *eXist* from the driver **Type** drop-down list.
5. Click the **Add** button to add the eXist driver files.

The following driver files should be added in the dialog box for setting up the eXist datasource. They are found in the installation directory of the eXist database server. Please make sure you copy the files from the installation of the eXist server where you want to connect from Oxygen XML Editor.

- exist.jar
- lib/core/xmldb.jar
- lib/core/xmlrpc-client-3.1.x.jar
- lib/core/xmlrpc-common-3.1.x.jar
- lib/core/ws-commons-util-1.0.x.jar
- lib/core/slf4j-api-1.x.x.jar (if available)
- lib/core/slf4j-log4j12-1.x.x.jar (if available)

The version number from the driver file names may be different for your eXist server installation.

6. Click the **OK** button to finish the connection configuration.

To watch our video demonstration about running XQuery against an eXist XML database, go to [http://www.oxygenxml.com/demo/eXist\\_Database.html](http://www.oxygenxml.com/demo/eXist_Database.html).

## How to Configure an eXist Connection

The steps for configuring a connection to an eXist database are as follows:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down list.
5. Enter the connection details.
  - a) Set the URI to the installed eXist engine in the **XML DB URI** field.

- b) Set the user name in the **User** field.
- c) Set the password in the **Password** field.
- d) Enter the start collection in the **Collection** field.

eXist organizes all documents in hierarchical collections. Collections are like directories. They are used to group related documents together. This text field allows the user to set the default collection name.

#### 6. Click the **OK** button to finish the connection configuration.

To watch our video demonstration about running XQuery against an eXist XML database, go to [http://www.oxygenxml.com/demo/eXist\\_Database.html](http://www.oxygenxml.com/demo/eXist_Database.html).

#### *The Create eXist-db XML Connection Dialog Box*

A quick way to create an eXist connection is to use the dedicated **Create eXist-db XML connection** dialog box. [Open the Preferences dialog box](#), go to **Data Sources** and click **Create eXist-db XML connection**. After you fill in the fields, click **OK** and go to **Window > Show View > Data Source Explorer** to view your connection.

To create an eXist connection using this dialog box, Oxygen XML Editor expects the `exist/webstart/exist.jnlp` path to be accessible at the provided **Host** and **Port**.

### How to Configure MarkLogic Support

Follow this procedure to configure the support for a MarkLogic database:

1. Download the MarkLogic driver from [MarkLogic Community site](#).
2. [Configure a MarkLogic Data Source driver](#).
3. [Configure a MarkLogic Connection](#).
4. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

### How to Configure a MarkLogic Data Source

Available in the Enterprise edition only.



**Note:** Oxygen XML Editor supports MarkLogic version 4.0 or later.

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *MarkLogic* from the driver **Type** drop-down list.
5. Click the **Add** button to add the MarkLogic driver file (`marklogic-xcc-{server_version}`, where `{server_version}` is the MarkLogic server version.)

You can download the driver file from: <http://community.marklogic.com/download>.

6. Click the **OK** button to finish the data source configuration.

### How to Configure a MarkLogic Connection

Available in the Enterprise edition only.



**Note:** Oxygen XML Editor supports MarkLogic version 4.0 or later.

The steps for configuring a connection to a MarkLogic database are as follows:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down list.
5. Enter the connection details.
  - a) The host name or IP address of the installed MarkLogic engine in the **XDBC Host** field.

Oxygen XML Editor uses XCC connector to interact with MarkLogic XDBC server and requires the basic authentication schema to be set. Starting with version MarkLogic 4.0 the default authentication method when you create a HTTP or WebDAV Server is digest, so make sure to change it to basic.

- b) Set the port number of the MarkLogic engine in the **Port** field. A MarkLogic XDBC application server must be configured on the server on this port. This XDBC server will be used to execute XQuery expressions against the server. Later, if you want to change the XDBC server, instead of editing the configuration just use the **Use it to execute queries** action from Data Source Explorer.
- c) Set the user name to access the MarkLogic engine in the **User** field.
- d) Set the password to access the MarkLogic engine in the **Password** field.
- e) Optionally set the URL used for browsing the MarkLogic database in the **Data Source Explorer** view in the **WebDAV URL** field.

The **Database** field specifies the database over which the XQuery expressions are executed. If you set this option to default, the database associated to the application server of the configured port is used.

6. Click the **OK** button to finish the connection configuration.

## How to Configure Documentum xDb (X-Hive/DB) 10 Support

Follow this procedure to configure the support for a Documentum xDb (X-Hive/DB) 10 database:

1. [Configure a Documentum xDb Data Source driver](#).
2. [Configure a Documentum xDb Connection](#).
3. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

## How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source

Available in the Enterprise edition only.

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select **XHive** from the driver **Type** drop-down list.
5. Click the **Add** button to add the XHive driver files.

The driver files for the Documentum xDb (X-Hive/DB) 10 database are found in the Documentum xDb (X-Hive/DB) 10 lib directory from the server installation folder:

- antlr-runtime.jar
- aspectjrt.jar
- icu4j.jar
- xhive.jar
- google-collect.jar

6. Click the **OK** button to finish the data source configuration.

## How to Configure an Documentum xDb (X-Hive/DB) 10 Connection

The steps for configuring a connection to a Documentum xDb (X-Hive/DB) 10 database are as follows:

 **Note:** The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Editor. The following procedure explains the `xhive://` protocol connection type.

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down list.
5. Enter the connection details.

- Set the URL property of the connection in the **URL** field.

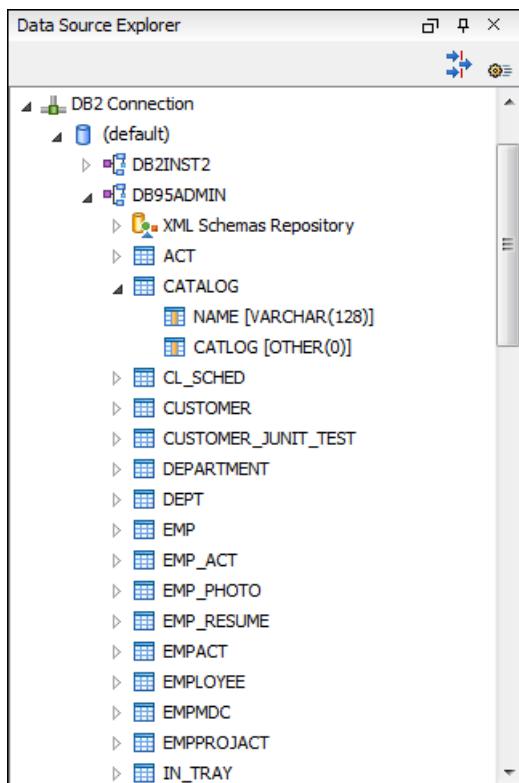
If the property is a URL of the form *xhive://host:port*, the Documentum xDb (X-Hive/DB) 10 connection will attempt to connect to a Documentum xDb (X-Hive/DB) 10 server running behind the specified TCP/IP port.

- Set the user name to access the Documentum xDb (X-Hive/DB) 10 engine in the **User** field.
- Set the password to access the Documentum xDb (X-Hive/DB) 10 engine in the **Password** field.
- Set the name of the database to access from the Documentum xDb (X-Hive/DB) 10 engine in the **Database** field.
- Check the **Run XQuery in read / write session (with committing)** checkbox if you want to end the session with a commit. Otherwise, the session ends with a rollback.

- Click the **OK** button to finish the connection configuration.

## Data Source Explorer View

The **Data Source Explorer** view displays your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded to the column level. supports multiple simultaneous database connections and the connection tree provides an easy method for browsing them.



**Figure 339: Data Source Explorer View**

The following objects are displayed in the **Data Source Explorer** view:

- **Connection**
- **Collection (Catalog)**
- **XML Schema Repository**
- **XML Schema Component**
- **Schema**
- **Table**
- **System Table**
- **Table Column**

A  **Collection** (called *catalog* in some databases) is a hierarchical container for resources and sub-collections. There are two types of resources:

-  **XML resource** - an XML document or document fragment, selected by a previously executed XPath query.
-  **non-XML resource** -any resource that is not recognized as XML.

 **Note:** For some connections you can add or move resources into a container by dragging them from:

- the **Project view**
- the default file system application (for example, Windows Explorer in Windows or Finder in Mac OS X)
- another database container

The following actions are available in the toolbar of this view:

 **Filters**

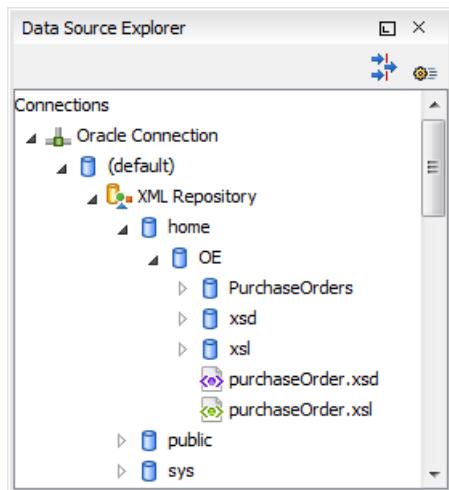
Opens the **Data Sources / Table Filters** [Preferences page](#), allowing you to decide which table types are displayed in the **Data Source Explorer** view.

 **Configure Database Sources**

Opens the **Data Sources** [preferences page](#) where you can configure both data sources and connections.

### Oracle XML DB Browser

Oracle XML DB is a feature of the Oracle database. It provides a high-performance, native XML storage and retrieval technology. Oxygen XML Editor allows you to browse the native Oracle XML Repository and perform various operations on the resources in the repository.



**Figure 340: Browsing the Oracle XML DB Repository**

The actions available at XML Repository level are as follows:

 **Refresh**

Performs a refresh of the XML Repository.

**Add container**

Adds a new child container to the XML Repository.

 **Add resource**

Adds a new resource to the XML Repository.

The actions available at container level are as follows:

## Refresh

Performs a refresh of the selected container.

## Add container

Adds a new child container to the current one.

## Add resource

Adds a new resource to the folder.

## Delete

Deletes the current container.

## Properties

Shows various properties of the current container.

The actions available at resource level are as follows:

## Refresh

Performs a refresh of the selected resource.

## Open

Opens the selected resource in the editor.

## Rename

Renames the current resource

## Move

Moves the current resource to a new container (also available through drag and drop).

## Delete

Deletes the current resource.

## Copy location

Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

## Properties

Shows various properties of the current resource.

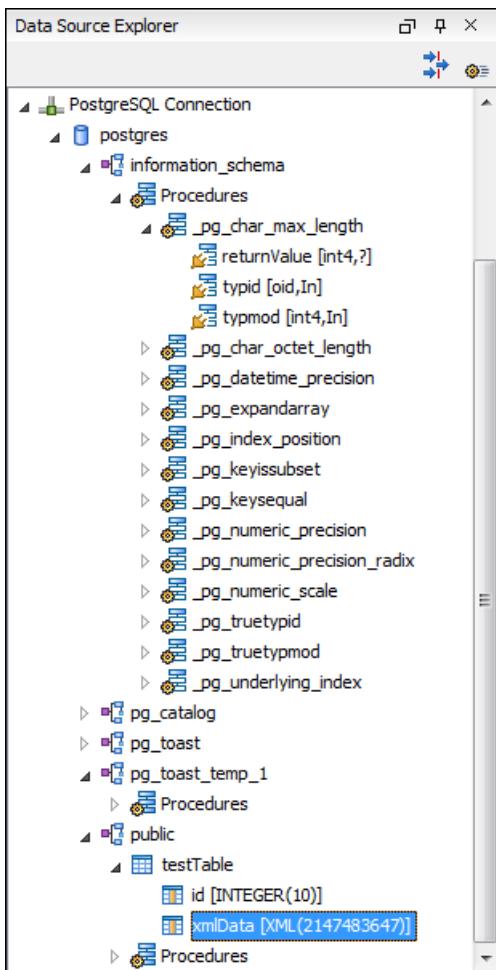
## Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

For running an XQuery transformation on collections from the XML Repository, please see [a tutorial from Oracle](#).

## PostgreSQL Connection

Oxygen XML Editor allows you to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.



**Figure 341: Browsing a PostgreSQL repository**

The actions available at container level are as follows:

#### C Refresh

Performs a refresh of the selected container.

The actions available at resource level are as follows:

#### C Refresh

Performs a refresh of the selected database table.

#### >Edit

Opens the selected database table in the **Table Explorer** view.

#### Export to XML ...

Exports the content of the selected database table as an XML file using *the dialog from importing data from a database*.

#### Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

### Berkeley DB XML Connection

This section explains the actions that are available on a Berkeley DB XML connection.

## Actions Available at Connection Level

In a Berkeley DB XML repository, the actions available at connection level in the **Data Source Explorer** view are as follows:

### Refresh

Performs a refresh for the sub-tree of the selected node.

### Disconnect

Closes the current database connection.

### Configure Database Sources

Opens [the Data Sources preferences page](#) where you can configure both data sources and connections.

### Add container

Adds a new container in the repository with the following attributes:

- **Name** - The name of the new container.
- **Container type** - At creation time, every container must have a type defined for it. This container type identifies how XML documents are stored in the container. As such, the container type can only be determined at container creation time. You cannot change it when subsequent container opens. Containers can have one of the following types specified for them:
  - **Node container** - XML documents are stored as individual nodes in the container. Each record in the underlying database contains a single leaf node, its attributes and attribute values (if any), and its text nodes (if any). Berkeley DB XML also keeps the information it requires to reassemble the document from the individual nodes stored in the underlying databases. This is the default, and preferred, container type.
  - **Whole document container** - The container contains entire documents. The documents are stored without any manipulation of line breaks or whitespace.
- **Allow validation** - If checked, it causes documents to be validated when they are loaded into the container. The default behavior is to not validate documents.
- **Index nodes** - If checked, it causes indices for the container to return nodes rather than documents. The default is to index at the document level. This property has no meaning if the container type is **Whole document container**.

### Properties

Shows a dialog box that contains a list of the Berkeley connection properties (*version, home location, default container type, compression algorithm, etc.*)

## Actions Available at Container Level

In a Berkeley DB XML repository, the actions available at container level in the **Data Source Explorer** view are as follows:

### Add Resource

Adds a new XML resource to the selected container.

### Rename

Allows you to specify a new name for the selected container.

### Delete

Removes the selected container from the database tree.

### Edit indices

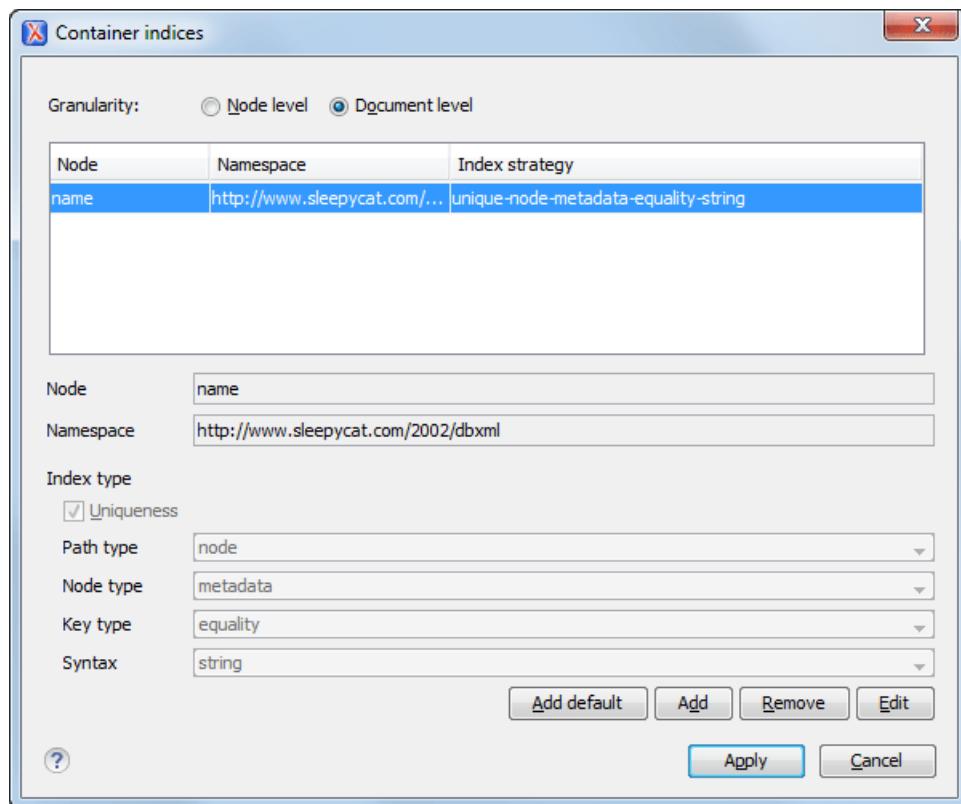
Allows you to edit the indices for the selected container.

### Refresh

Performs a refresh for the sub-tree of the selected node.

## Properties

Displays a dialog box with a list of properties of the Berkeley container (such as *container type*, *auto indexing*, *page size*, *validate on load*, *compression algorithm*, *number of documents*, etc.)



**Figure 342: Container indices**

The fields of the dialog box are as follows:

- Granularity:
  - **Document level** - Good option for retrieving large documents.
  - **Node level** - Good option for retrieving nodes from within documents.
- Add / Edit indices:
  - **Node** - The node name.
  - **Namespace** - The index namespace.
- Index strategy:
  - **Index type:**
    - **Uniqueness** - Indicates whether the indexed value must be unique within the container.
    - **Path type:**
      - **node** - Indicates that you want to index a single node in the path.
      - **edge** - Indicates that you want to index the portion of the path where two nodes meet.
  - **Node type:**
    - **element** - An element node in the document content.
    - **attribute** - An attribute node in the document content.
    - **metadata** - A node found only in the metadata content of a document.
  - **Key type:**

- **equality** - Improves the performances of tests that look for nodes with a specific value.
- **presence** - Improves the performances of tests that look for the existence of a node regardless of its value.
- **substring** - Improves the performance of tests that look for a node whose value contains a given sub-string.
- **Syntax types** - The syntax describes the type of data the index contains and is mostly used to determine how indexed values are compared.

## Actions Available at Resource Level

In a Berkeley DB XML repository, the actions available at resource level in the **Data Source Explorer** view are as follows:

### Refresh

Performs a refresh of the selected resource.

### Open

Opens the selected resource in the editor.

### Rename

Allows you to change the name of the selected resource.

### Move

Allows you to move the selected resource in a different container in the database tree (also available through drag and drop).

### Delete

Removes the selected resource from the container.

### Copy location

Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

### Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

## eXist Connection

This section explains the actions that are available on an eXist connection.

## Actions Available at Connection Level

For an eXist database, the actions available at connection level in the **Data Source Explorer** view are as follows:

### Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

### Disconnect

Closes the current database connection.

### Refresh

Performs a refresh for the sub-tree of the selected node.

## Actions Available at Container Level

For an eXist database, the actions available at container level in the **Data Source Explorer** view are as follows:

### New File

Creates a file in the selected container.

### New Collection

Creates a collection.

## Import Folders

Adds the content of specified folders from the local file system.

### Import Files

Adds a set of XML resources from the local file system.

## Cut

Cuts the selected containers.

## Copy

Copies the selected containers.

-  **Note:** You can add or move resources into the container by dragging them from the **Project** view, the default file management application (for example, Windows Explorer on Windows or Finder on OS X), or from another database container.

## Paste

Paste resources into the selected container.

## Rename

Allows you to change the name of the selected collection.

### Delete

Removes the selected collection.

### Refresh

Performs a refresh of the selected container.

## Properties

Allows you to view various useful properties associated with the container, such as *name*, *creation date*, *owner*, *group*, or *permissions*.

## Actions Available at Resource Level

For an eXist database, the actions available at resource level in the **Data Source Explorer** view are as follows:

### Refresh

Performs a refresh of the selected resource.

## Open

Opens the selected resource in the editor.

## Rename

Allows you to change the name of the selected resource.

## Cut

Cuts the selected resources.

## Copy

Copies the selected resources.

-  **Note:** You can add or move resources into the container by dragging them from the **Project** view, the default file management application (for example, Windows Explorer on Windows or Finder on OS X), or from another database container.

## Paste

Pastes the copied resources.

### Delete

Removes the selected resource from the collection.

### Copy location

Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

### Properties

Allows you to view various useful properties associated with the resource.

### Save As

Allows you to save the name of the selected binary resource as a file on disk.

### Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

## MarkLogic Connection

Once you configure a MarkLogic connection, you can use the **Data Source Explorer** view to display all the application servers that are configured on the server. You can expand each application server and view all the modules that it is configured to use. The **Data Source Explorer** view allows you to open and edit these modules.



**Note:** To browse modules located in a database, directory properties must be associated with them. These directory properties are generated automatically if the *directory creation* property of the database is set to automatic. If this property is set to *manual* or *manual-enforced*, add the directory properties of the modules manually, using the XQuery function `xdmp:directory-create()`.

### Manually Adding Directory Properties

For two documents with the `/code/modules/main.xqy` and `/code/modules/imports/import.xqy` IDs, run this query:  
`(xdmp:directory-create('/code/modules/'),  
xdmp:directory-create('/code/modules/imports/'))`.

For further information about directory properties go to: <http://blakeley.com/blogofile/2012/03/19/directory-assistance/>.

When you execute or debug XQuery files opened from this view, the imported modules are better identified by the MarkLogic server. In a module, you are also able to add breakpoints that the debugger takes into account.

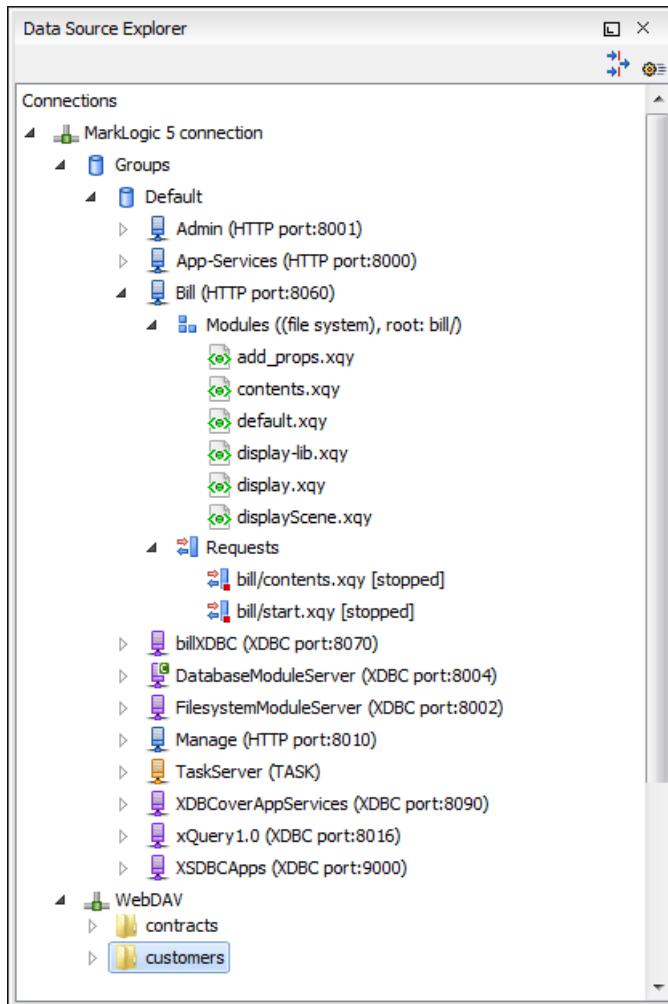


**Note:** Add breakpoints in the modules of the application server that executes the debugging.



**Note:** Open XQuery modules from the application server involved in the debugging or execution process.

In the **Requests** container of each application server, Oxygen XML Editor displays both the queries that are stopped for debugging purposes and the queries that are still running. To clean up the entire **Requests** container at the end of your session, right-click it and use the **Cancel all running requests** action.



**Figure 343: MarkLogic Connection in Data Source Explorer**

The **Data Source Explorer** view displays all the application servers available on the MarkLogic server. To change the XDBC application server that Oxygen XML Editor uses to execute XQuery expressions, select the **Use it to execute queries** option from its contextual menu.

To manage resources for a MarkLogic database through WebDAV, configure a WebDAV URL in [the MarkLogic connection](#).

The following actions are available in the contextual menu of the WebDAV connection:

- Connection level actions:
  - ⚙️ **Configure Database Sources...**  
Opens the **Data Sources** [preferences page](#). Here you can configure both data sources and connections.
  - New Folder...**  
Creates a new folder on the server.
  - Import Files...**  
Allows you to add a new file on the server.
  - Refresh**  
Performs a refresh of the connection.
  - Find/Replace in Files...**  
Allows you to find and replace text in multiple files from the server.

- Folder level actions:

**New File**

Creates a new file on the server in the current folder.

**New Folder...**

Creates a new folder on the server.

**Import Folders...**

Imports folders on the server.

 **Import Files**

Allows you to add a new file on the server in the current folder.

 **Cut**

Removes the current selection and places it in the clipboard.

 **Copy**

Copies the current selection into the clipboard.

**Rename**

Allows you to change the name of the selected folder.

 **Delete**

Removes the selected folder.

 **Refresh**

Refreshes the sub-tree of the selected node.

 **Find/Replace in Files...**

Allows you to find and replace text in multiple files from the server.

- File level actions:

 **Open**

Allows you to open the selected file in the editor.

 **Cut**

Removes the current selection and places it in the clipboard.

 **Copy**

Copies the current selection into the clipboard.

**Copy Location**

Copies an application-specific URL for the selected resource into the clipboard. You can use this URL for various actions, such as opening or transforming the resources.

**Rename**

Allows you to change the name of the selected file.

 **Delete**

Removes the selected file.

 **Refresh**

Performs a refresh of the selected node.

 **Properties**

Displays the properties of the current file in a **Properties** dialog box.

 **Find/Replace in Files...**

Allows you to find and replace text in multiple files from the server.

## Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

## Documentum xDb (X-Hive/DB) Connection

This section explains the actions that are available on a Documentum xDb (X-Hive/DB) 10 connection.

### Actions Available at Connection Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at connection level in the **Data Source Explorer** view are as follows:

#### Refresh

Performs a refresh for the sub-tree of the selected node.

#### Disconnect

Closes the current database connection.

#### Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

#### Add library

Allows you to add a new library.

#### Insert XML Instance

Allows you to add a new XML resource directly into the database root. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.

#### Insert non-XML Instance

Allows you to add a new non-XML resource directly in the database root.

#### Properties

Displays the connection properties.

### Actions Available at Catalog Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at catalog level in the **Data Source Explorer** view are as follows:

#### Refresh

Performs a refresh of the selected catalog.

#### Add as models

Allows you to add a new abstract schema model to the selected catalog.

#### Set default schema

Allows you to set a default DTD to be used for parsing. It is not possible to set a default XML Schema.

#### Clear default schema

Allows you to clear the default DTD. The action is available only if there is a DTD set as default.

#### Properties

Displays the catalog properties.

### Actions Available at Schema Resource Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at schema resource level in the **Data Source Explorer** view are as follows:

#### Refresh

Performs a refresh of the selected schema resource.

#### Open

Opens the selected schema resource in the editor.

**Rename**

Allows you to change the name of the selected schema resource.

**Save As**

Allows you to save the selected schema resource as a file on disk.

**✖ Delete**

Removes the selected schema resource from the catalog.

**Copy location**

Allows you to copy the URL of the selected schema resource to the clipboard.

**Set default schema**

Allows you to set the selected DTD to be used as default for parsing. The action is available only for DTD.

**Clear default schema**

Allows you to unset the selected DTD. The action is available only if the selected DTD is the current default to be used for parsing.

**Actions Available at Library Level**

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at library level in the **Data Source Explorer** view are as follows:

**↻ Refresh**

Performs a refresh of the selected library.

**Add library**

Adds a new library as a child of the selected library.

**Add local catalog**

Adds a catalog to the selected library. By default, only the root-library has a catalog, and all models are stored there.

**➕ Insert XML Instance**

Allows you to add a new XML resource to the selected library. See [Documentum xDb \(X-Hive/DB\) 10 Parser Configuration](#) for more details.

**➕ Insert non-XML Instance**

Allows you to add a new non-XML resource to the selected library.

**Rename**

Allows you to specify a new name for the selected library.

**Move**

Allows you to move the selected library to a different one (also available through drag and drop).

**✖ Delete**

Removes the selected library.

**Properties**

Displays the library properties.

**Actions Available at Resource Level**

When an XML instance document is added for a Documentum xDb (X-Hive/DB) 10 database, the actions available at resource level in the **Data Source Explorer** view are as follows:

**↻ Refresh**

Performs a refresh of the selected resource.

**Open**

Opens the selected resource in the editor.

**Rename**

Allows you to change the name of the selected resource.

**Move**

Allows you to move the selected resource into a different library in the database tree (also available through drag and drop).



**Note:** You can copy or move resources by dragging them from another database catalog.

**Save As**

Allows you to save the selected binary resource as a file on disk.

**✖ Delete**

Removes the selected resource from the library.

**Copy location**

Allows you to copy the URL of the selected resource to the clipboard.

**Add AS model**

Allows you to add an XML schema to the selected XML resource.

**Set AS model**

Allows you to set an active AS model for the selected XML resource.

**Clear AS model**

Allows you to clear the active AS model of the selected XML resource.

**Properties**

Displays the resource properties. Available only for XML resources.

**Compare**

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

Validation of an XML resource stored in an Documentum xDb (X-Hive/DB) 10 database is done against the schema associated with the resource in the database.

**Documentum xDb (X-Hive/DB) 10 Parser Configuration for Adding XML Instances**

When an XML instance document is added to a Documentum xDb (X-Hive/DB) 10 connection or library, it is parsed with an internal XML parser of the database server. The following options are available for configuring this parser:

- DOM Level 3 parser configuration parameters. More about each parameter can be found here: [DOM Level 3 Configuration](#).
- Documentum xDb (X-Hive/DB) 10 specific parser parameters (for more information please consult the Documentum xDb (X-Hive/DB) 10 manual):
  - **xhive-store-schema** - If checked, the corresponding DTD or XML schemas are stored in the catalog during validated parsing.
  - **xhive-store-schema-only-internal-subset** - Stores only the internal sub-set of the document (not an external sub-set). This option modifies the **xhive-store-schema** (only has a function when that parameter is set to true, and when a DTD is involved). Select this option if you only want to store the internal sub-set of the document (not the external sub-set).
  - **xhive-ignore-catalog** - Ignores the corresponding DTD and XML schemas in the catalog during validated parsing.
  - **xhive-psvi** - Stores **psvi** information on elements and attributes. Documents parsed with this feature turned on give access to **psvi** information and enable support of data types by XQuery queries.
  - **xhive-sync-features** - With this convenience setting turned on, parameter settings of `XhiveDocumentIf` are synchronized with the parameter settings of `LSParser`. Note that parameter settings **xhive-psvi** and **schema-location** are always synchronized.

## Troubleshooting

*Cannot save the file. DTD factory class org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl does not extend from DTDDVFactory*

I am able to access my XML Database in the Data Source Explorer and open files for reading but when I try to save changes to a file back into the database, I receive the following error: "Cannot save the file. DTD factory class org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl does not extend from DTDDVFactory." How can I fix this?

### Answer:

xhive.jar contains a MANIFEST.MF with a classpath:

```
Class-Path: core/antlr-runtime.jar core/aspectjrt.jar core/fastutil-shrinked.jar
core/google-collect.jar core/icu4j.jar core/lucene-regex.jar core/lucene.jar
core/serializer.jar core/xalan.jar core/xercesImpl.jar
```

Because the driver was configured to use xhive.jar directly from the xDB installation (where many other jars are located), core/xercesImpl.jar from the xDB installation directory is loaded even though it is not specified in the list of jars from the data source driver configuration (it is in the classpath from xhive.jar's *MANIFEST.MF*). A simple workaround for this issue is to copy **ONLY** the jar files used in the driver configuration to a separate folder and configure the data source driver to use them from there.

## XQuery and Databases

---

XQuery is a native XML query language that is useful for querying XML views of relational data to create XML results. It also provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data. The following database systems supported in Oxygen XML Editor offer XQuery support:

- *Native XML Databases:*
  - Berkeley DB XML
  - eXist
  - MarkLogic (validation support available starting with version 5)
  - Documentum xDb (X-Hive/DB) 10
- *Relational Databases:*
  - IBM DB2
  - Microsoft SQL Server (validation support not available)
  - Oracle (validation support not available)

## Build Queries with Drag and Drop from the Data Source Explorer View

When a query is edited in the XQuery editor, the XPath expressions can be composed quickly by dragging them from the **Data Source Explorer** view and dropping them into the editor panel.

1. [Configure the data source](#) to the relational database.
2. [Configure the connection](#) to the relational database.
3. Browse the connection in the **Data Source Explorer** view, expanded to the table or column that you want to insert in the query.
4. Drag the table or column name to the XQuery editor panel.
5. Drop the table or column name where the XPath expression is needed.

An XPath expression that selects the dragged name is inserted in the XQuery document at the caret position.

## XQuery Transformation

XQuery is designed to retrieve and interpret XML data from any source, whether it is a database or document. Data is stored in relational databases but it is often required that the data be extracted and transformed as XML when interfacing to other components and services. Also, it is an XPath-based querying language supported by most NXD vendors. To perform a query, you need an XQuery transformation scenario.

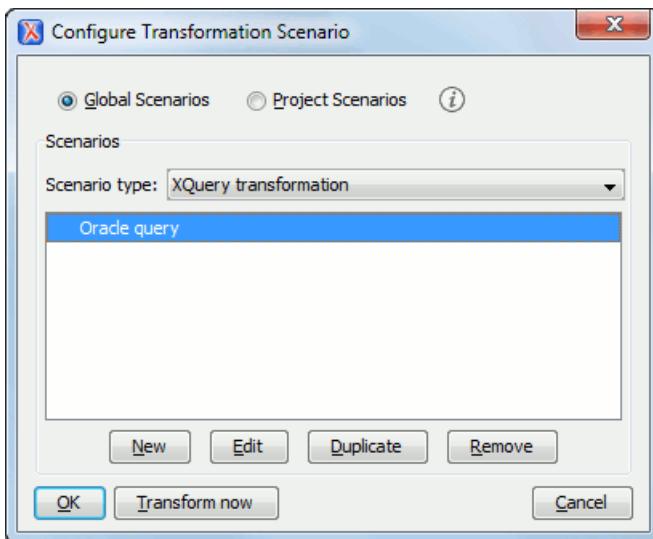
1. Configure a data source for the database.

The data source can be *relational* or *XML native*.

2. Configure an XQuery transformation scenario.

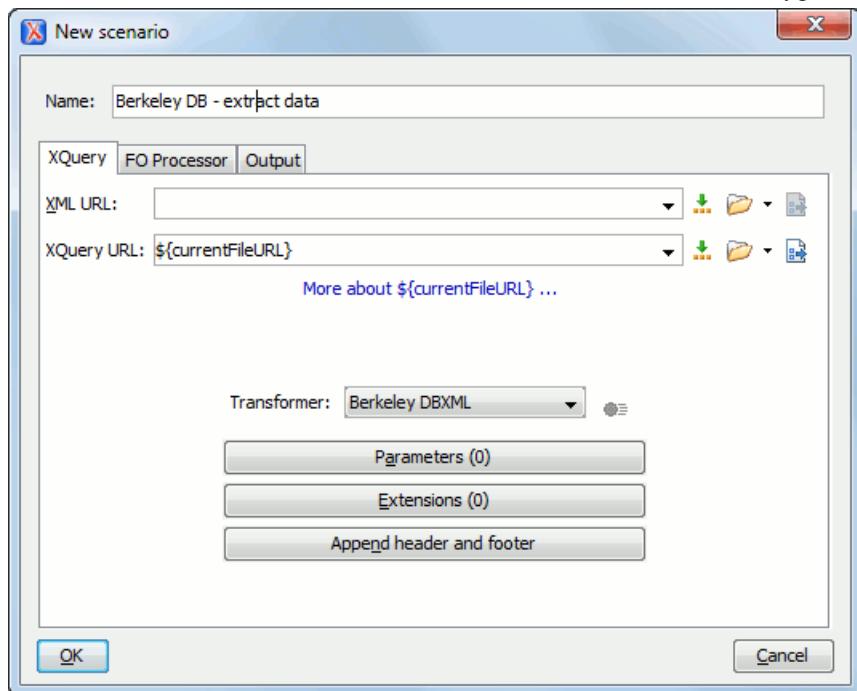
- a) Click the  **Configure Transformation Scenario** toolbar button or go to menu **Document > Transformation > Configure Transformation Scenario**.

The **Configure Transformation Scenario** dialog box is opened.



- b) Click the **New** button in the dialog box.

The dialog box for editing an XQuery scenario is opened.



**Figure 344: Edit Scenario Dialog Box**

- c) Insert the scenario name in the dialog box for editing the scenario.
- d) Choose the database connection in the **Transformer** drop-down list.
- e) Configure any other parameters as needed.

For an XQuery transformation, the output tab has an option called **Sequence** that allows you to execute an XQuery in lazy mode. The amount of data extracted from the database is controlled from the option [Size limit on Sequence view](#). If you choose **Perform FO Processing** in the **FO Processor** tab, the **Sequence** option is ignored.

- f) Click the **OK** button to finish editing the scenario.

Once the scenario is associated with the XQuery file, the query can include calls to specific XQuery functions that are implemented by that engine. The available functions depend on the target database engine selected in the scenario. For example, for eXist and Berkeley DB XML, [the Content Completion Assistant](#) lists the functions supported by that database engine. This is useful for only inserting calls to the supported functions (standard XQuery functions or extension ones) into the query .

 **Note:** An XQuery transformation is executed against a Berkeley DB XML server as a transaction using the query transaction support of the server.

### 3. Run the scenario.

To view a more complex value returned by the query that cannot be entirely displayed in the XQuery query result table at the bottom of the Oxygen XML Editor window (for example, an XMLTYPE or CLOB value), do the following:

- Right-click on that table cell.
- Select the **Copy cell** action from the pop-up menu to copy the value into the clipboard.
- Paste the value wherever you need it (for example, in an opened XQuery editor panel of Oxygen XML Editor).

## XQuery Database Debugging

This section describes the procedures for debugging XQuery transformations that are executed against MarkLogic and Berkeley DB XML databases.

## Debugging with MarkLogic

To start a debug session against the MarkLogic engine, configure a [MarkLogic data source](#) and a [MarkLogic connection](#). Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Editor accesses. On the server side, debugging must be activated in the XDBC server and in the *Task Server* section of the server control console (the switch *debug allow*). If the debugging is not activated, the MarkLogic server reports the `DBG-TASKDEBUGALLOW` error.

The MarkLogic XQuery debugger integrates seamlessly into the [XQuery Debugger perspective](#). If you have a MarkLogic scenario configured for the XQuery file, you can choose to [debug the scenario](#) directly. If not, switch to the XQuery Debugger perspective, open the XQuery file in the editor, and select the MarkLogic connection in the XQuery engine selector from the [debug control toolbar](#). For general information about how a debugging session is started and controlled see the [Working with the Debugger](#) section.

If you want to debug an XQuery file stored on the MarkLogic server, we recommend you to use the **Data Source Explorer** view to open the module and start the debugging session. This improves the resolving of any imported modules.

Oxygen XML Editor supports collaborative debugging. This feature allows multiple users to participate in the same debugging session. You can start a debugging session and at a certain point another user can continue it.

In a MarkLogic debugging session, when you add a breakpoint on a line where the debugger never stops, Oxygen XML Editor displays a warning message. These warnings are displayed for breakpoints you add either in the main XQuery (which you can open locally or from the server), or for breakpoints you add in any XQuery that is opened from the connection that participates in the debugging session.

To watch our video demonstration about the XQuery debugger for MarkLogic, go to <http://oxygenvml.com/demo/XQueryDebuggerforMarkLogic.html>.

## Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is available only for MarkLogic server versions 4.0 or newer.
- For MarkLogic server versions 4.0 or newer, there are three XQuery syntaxes that are supported: '0.9-ml' (inherited from MarkLogic 3.2), '1.0-ml', and '1.0'.
- The local debugger user interface presents all the debugging steps that the MarkLogic server executes and the results or possible errors of each step.
- All declared variables are presented as strings. The **Value** column of the **Variables** view contains the expression from the variable declaration. It can be evaluated by copying the expression with the **Copy value** action from the contextual menu of [the Variables view](#) and pasting it in [the XWatch view](#).
- There is no support for [Output to Source Mapping](#).
- There is no support for [showing the trace](#).
- You can set [Breakpoints](#) in imported modules in one of the following cases:
  - when you open the module from the context of the application server involved in the debugging, using the **data source explorer**
  - when the debugger automatically opens the modules in the Editor
- No breakpoints are set in modules from the same server that are not involved in the current debugging session.
- No support for [profiling](#) when an XQuery transformation is executed in the debugger.

## Debugging Queries Which Import Modules

When debugging queries on a MarkLogic database that imports modules stored in the database, the recommended steps for placing a *breakpoint* in a module are as follows:

1. Start the debugging session with the action  **Debug Scenario** from the **Transformation** toolbar or the  **XQuery Debugger** toolbar button.
2. Click  **Step into** repeatedly until reaching the desired module.
3. Add the module to the current [project](#) for easy access.
4. Set breakpoints in the module as needed.

5. *Continue debugging* the query.

When starting a new debugging session, make sure that the modules that you will debug are already opened in the editor. This is necessary so that the breakpoints in the modules will be considered. Also, make sure that there are no other opened modules that are not involved in the current debugging session.

### Debugging with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is supported in the Oxygen XML Editor XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic debugger.

## WebDAV Connection

---

This section explains how to work with a WebDAV connection in the **Data Source Explorer** view.

### How to Configure a WebDAV Connection

By default, Oxygen XML Editor is configured to contain a WebDAV data source connection called **WebDAV (S)FTP**. Based on this data source, you can create a WebDAV connection for browsing and editing data from a database that provides a WebDAV interface. The connection is available in *the Data Source Explorer view*. The steps for configuring a WebDAV connection are as follows:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the WebDAV data sources in the **Data Source** drop-down list.
5. Enter the connection details:
  - a) Set the URL to the WebDAV repository in the field **WebDAV URL**.
  - b) Set the user name that is used to access the WebDAV repository in the **User** field.
  - c) Set the password that is used to access the WebDAV repository in the **Password** field.
6. Click the **OK** button.

To watch our video demonstration about the WebDAV support in Oxygen XML Editor, go to [http://www.oxygenxml.com/demo/WebDAV\\_Support.html](http://www.oxygenxml.com/demo/WebDAV_Support.html).

## WebDAV Connection Actions

This section explains the actions that are available for a WebDAV connection in the **Data Source Explorer** view.

### Actions Available at Connection Level

The contextual menu of a WebDAV connection in the **Data Source Explorer** view contains the following actions:

#### **Configure Database Sources...**

Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

#### **Disconnect**

Stops the connection.

#### **Import Files...**

Allows you to add a new file on the server.

#### **New Folder...**

Creates a new folder on the server.

#### **Refresh**

Performs a refresh of the connection.

 **Find/Replace in Files...**

Allows you to find and replace text in multiple files from the server.

### **Actions Available at Folder Level**

The contextual menu of a folder node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

#### **New File**

Creates a new file on the server in the current folder.

#### **New Folder...**

Creates a new folder on the server.

#### **Import Folders...**

Imports folders on the server.

 **Import Files**

Allows you to add a new file on the server in the current folder.

 **Cut**

Removes the current selection and places it in the clipboard.

 **Copy**

Copies the current selection into the clipboard.

 **Paste**

Pastes the copied selection.

#### **Rename**

Allows you to change the name of the selected folder.

 **Delete**

Removes the selected folder.

 **Refresh**

Refreshes the sub-tree of the selected node.

 **Find/Replace in Files...**

Allows you to find and replace text in multiple files from the server.

### **Actions Available at File Level**

The contextual menu of a file node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

 **Open**

Allows you to open the selected file in the editor.

 **Cut**

Removes the current selection and places it in the clipboard.

 **Copy**

Copies the current selection into the clipboard.

#### **Copy Location**

Copies an application-specific URL for the selected resource into the clipboard. You can use this URL for various actions, such as opening or transforming the resources.

#### **Rename**

Allows you to change the name of the selected file.

### Delete

Removes the selected file.

### Refresh

Performs a refresh of the selected node.

### Properties

Displays the properties of the current file in a **Properties** dialog box.

### Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

---

## BaseX Support

This section explains how to configure the BaseX XML database support. The BaseX support is composed of two parts:

- Resource management in the **Data Source Explorer** view.
- XQuery execution.

### Resource Management

Resource management is available by creating a WebDAV connection to the BaseX server.

First of all, make sure the BaseX HTTP Server is started. For details about starting the BaseX HTTP server, go to [http://docs.basex.org/wiki/Startup#BaseX\\_HTTP\\_Server](http://docs.basex.org/wiki/Startup#BaseX_HTTP_Server). The configuration file for the HTTP server is named `.basex` and is located in the BaseX installation directory. This file helps you to find out the port on which the HTTP server is running. The default port for BaseX WebDAV is 8984.

To ensure that everything is functioning, open a WebDAV URL inside a browser and check to see if it works. For example, the following URL retrieves a document from a database named TEST:

`http://localhost:8984/webdav/TEST/etc/factbook.xml`.

Once you are sure that the BaseX WebDAV service is working, you can configure the WebDAV connection in Oxygen XML Editor as described in [How to Configure a WebDAV Connection](#) on page 840. The WebDAV URL should resemble this: `http://{hostname}:{port}/webdav/`. If the BaseX server is running on your own machine and it has the default configuration, the data required by the WebDAV connection is:

- WebDAV URL: `http://localhost:8984/webdav`
- User: admin
- Password: admin

Once the WebDAV connection is created, you can start browsing using [the Data Source Explorer view](#).

### XQuery Execution

XQuery execution is possible through an XQJ connection.

#### BaseX XQJ Data Source

First of all, create an XQJ data source as described in [How to Configure an XQJ Data Source](#) on page 763. The BaseX XQJ API-specific files that must be added in the configuration dialog are `xqj-api-1.0.jar`, `xqj2-0.1.0.jar` and `basex-xqj-1.2.3.jar` (the version names of the JAR file may differ). These libraries can be downloaded from [xqj.net/basex/basex-xqj-1.2.3.zip](http://xqj.net/basex/basex-xqj-1.2.3.zip). As an alternative, you can also find the libraries in the BaseX installation directory, in the `lib` sub-directory.

## BaseX XQJ Connection

The next step is to create an XQJ connection as described in [How to Configure an XQJ Connection](#) on page 763.

For a default BaseX configuration, the following connection details apply (you can modify them when necessary):

- *Port*: 1984
- *serverName*: localhost
- *user*: admin
- *password*: admin

## XQuery Execution

Now that the XQJ connection is configured, open the XQuery file you wish to execute in Oxygen XML Editor and create a *Transformation Scenario* as described in [XQuery Transformation](#) on page 713. In the **Transformer** drop-down list, select the name of the XQJ connection you created. Apply the transformation scenario and the XQuery will be executed.



---

# Chapter

# 17

---

## Importing Data

---

**Topics:**

- [\*Introduction\*](#)
- [\*Import from Database\*](#)
- [\*Import from MS Excel Files\*](#)
- [\*Import from HTML Files\*](#)
- [\*Import from Text Files\*](#)
- [\*Import Content Dynamically\*](#)

This chapter describes how you can import data stored in text format, Excel sheet, or relational database tables, into XML documents.

## Introduction

Computer systems and databases contain data in incompatible formats and one of the most time-consuming activities has been to exchange data between these systems. Converting the data to XML can greatly reduce complexity and create data that can be read by different types of applications.

This is why Oxygen XML Editor offers support for importing text files, MS Excel files, Database Data, and HTML files into XML documents. The XML documents can be further converted into other formats using the [Transform features](#).

## Import from Database

This section explains how to import data from a database into Oxygen XML Editor.

### Import Table Content as XML Document

The steps for importing the data from a relational database table are the following:

1. Go to menu **File > Import > Database Data...**

Clicking this action opens a dialog box with all the defined database connections:

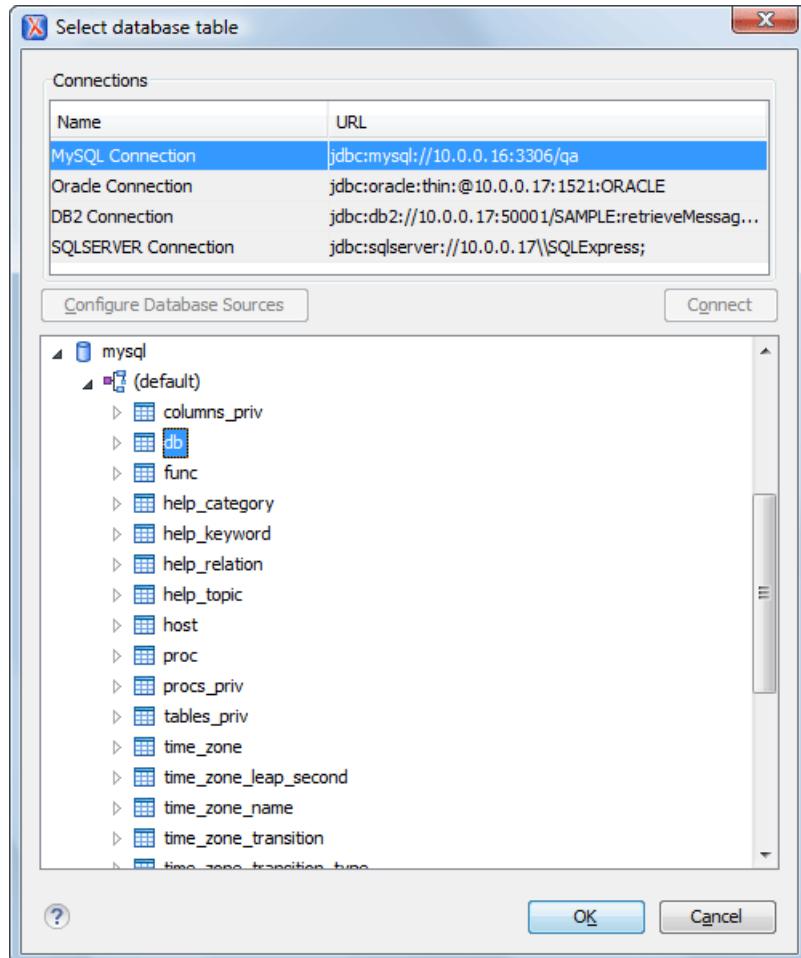


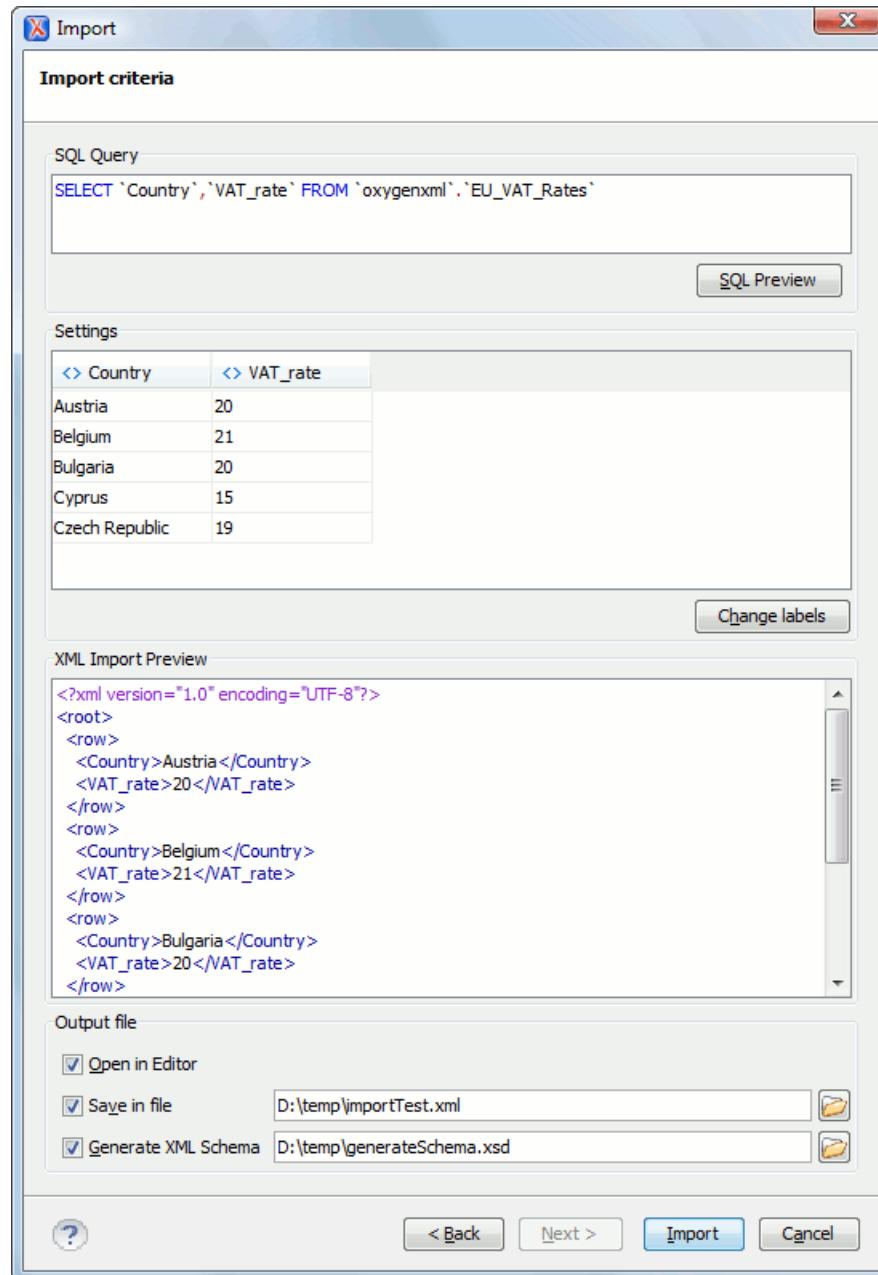
Figure 345: Select Database Table Dialog Box

2. Select the connection to the database that contains the data.

Only connections configured in relational data sources can be used to import data.

3. If you want to edit, delete, or add a data source or connection, click on the **Configure Database Sources** button. The **Preferences/Data Sources** option page is opened.
4. Click **Connect**.
5. From the catalogs list, click on a schema and choose the required table.
6. Click the **OK** button.

The **Import Criteria** dialog box opens with a default query string in the **SQL Query** pane:



**Figure 346: Import from Database Criteria Dialog Box**

The dialog box contains the following options:

- **SQL Preview** - If the **SQL Preview** button is pressed, the **Settings** pane displays the labels that are used in the XML document and the first five lines from the database. By default, all data items in the input are converted to element content, but this can be overridden by clicking the individual column headers. Clicking once on a column header causes the data from this column to be used as attribute values of the row elements. Click a second time

and the data from that column is ignored when generating the XML file. You can cycle through these options by continuing to click the column header. The following symbols are used in the column header to indicate the type of content the column is converted to:

- <> - data columns converted to element content
  - = - data columns converted to attribute content
  - x - ignored data
- **Change labels** - This button opens a new dialog box that allows you to edit the names of the root and row elements, change the XML name, and change the conversion criterion. The XML names can be edited by double-clicking the desired item and entering the required label. The conversion criterion can also be modified by selecting **ELEMENT**, **ATTRIBUTE**, or **SKIPPED** from the drop-down list.
  - **Open in editor** - If checked, the new XML document that is created from the imported text file, is opened in the editor.
  - **Save in file** - If checked, the new XML document is saved at the specified path.



**Note:** If this option is unchecked, while **Open in editor** is checked, the newly created document is opened in the editor as an unsaved file.

- **Generate XML Schema** - Allows you to specify the path of the generated XML Schema file.

## 7. Click the **SQL Preview** button.

The **SQL Query** string is editable. You can specify which fields are considered.

Use aliases if the following statements are true:

- the query string represents a join operation of two or more tables
- columns that are selected from different tables have the same name

The use of aliases avoids the confusion of two columns being mapped to the same name in the result document of the importing operation.

```
select s.subcat_id,
 s.nr as s_nr,
 s.name,
 q.q_id,
 q.nr as q_nr,
 q.q_text
 from faq.subcategory s,
 faq.question q
 where ...
```

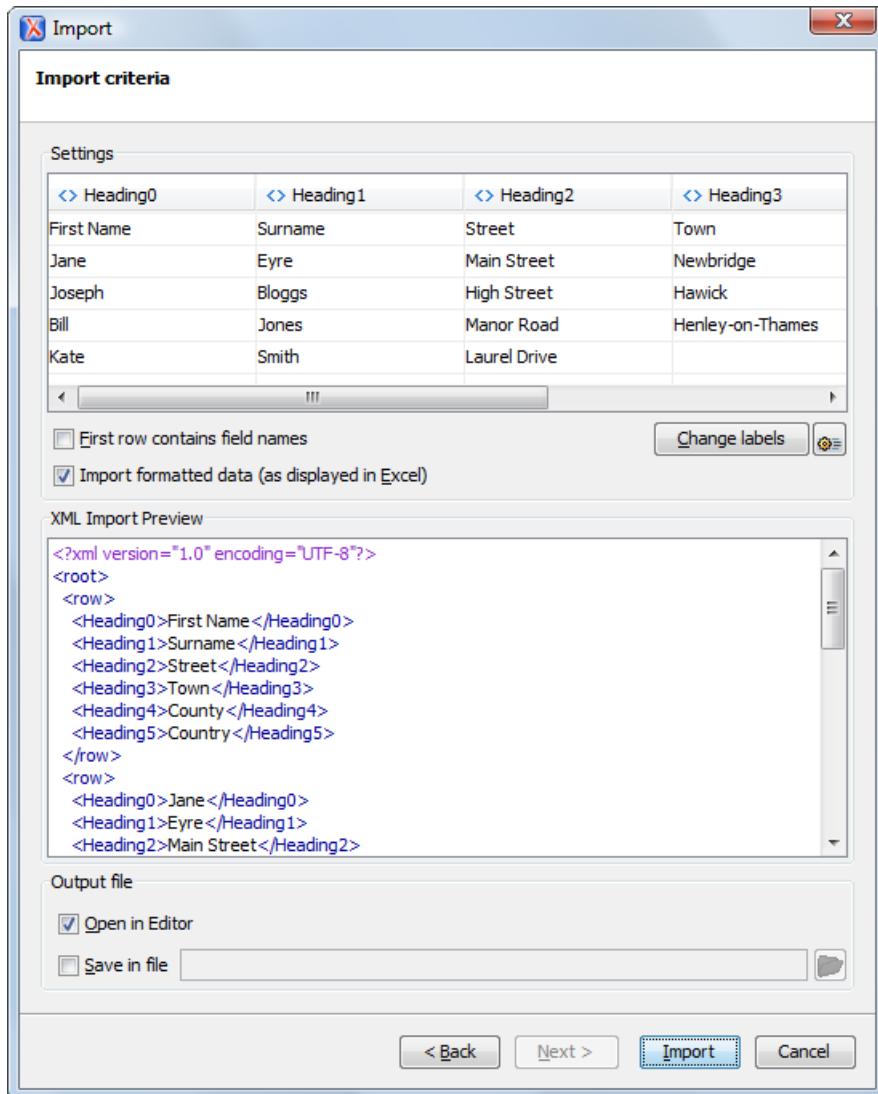
The input data is displayed in tabular form in the **Settings** pane. The **XML Import Preview** pane contains an example of what the generated XML looks like.

## Convert Table Structure to XML Schema

The structure of a table from a relational database can be imported in Oxygen XML Editor as an XML Schema. This feature is activated by the **Generate XML Schema** option from the **Import criteria** dialog box used in [the procedure for importing table data](#) as an XML instance document.

## Import from MS Excel Files

Oxygen XML Editor offers support for importing MS Excel Files. To import Excel files, go to **File > Import > MS Excel file** and in the **Import** dialog box select the file you want to import. In the **Available Sheets** section of this dialog box, the sheets of the document you are importing are presented. Select a sheet and click next to move on to the second **Import** dialog box.



**Figure 347: The "Import" Dialog Box - Import Criteria**

The **Settings** section presents the data from the Excel sheet in a tabular form. It also contains the following options:

- **First row contains field names** - Uses the content from the first row to name the columns.
- **Import formatted data (as displayed in Excel)** - Keeps the Excel styling.
- **Change labels** - Opens the **Presentation Names** dialog box. The following options are available:
  - **Root Element** - allows you to edit the name of the Root element.
  - **Row Element** - allows you to edit the name of the Row element.
  - **Real Name** - contains the original name of each Heading.
  - **XML Name** - allows you to modify the names of the Headings.
  - **Criterion** - allows you to transform the Heading elements to attributes of the Root element.
- **Import settings** - Opens the XML / Import preferences page.

The **XML Import Preview** section displays the Excel document in an XML format.

The **Output File** section contains the following options:

- **Open in Editor** - Opens the imported document in the Editor;
- **Save in File** - Saves the imported document in the specified location.

When you finish configuring the options in these dialog boxes, click **Import**.

## Import from MS Excel 2007-2010 (.xlsx)

To import XML from Excel 2007-2010 (.xlsx) documents, Oxygen XML Editor needs additional libraries from the release 3.10 of Apache POI project. Follow these steps:

1. Download version 3.10 of the Apache POI project from <http://poi.apache.org/download.html>. If a newer version has been released, you can still find version 3.10 at <http://archive.apache.org/dist/poi/release/bin/>.
2. From the downloaded project locate and add the following .jar files in the lib directory of the installation folder of Oxygen XML Editor :
  - dom4j-1.6.1.jar
  - poi-ooxml-3.10-FINAL-20140208.jar
  - poi-ooxml-schemas-3.10-FINAL-20140208.jar
  - xmlbeans-2.3.0.jar

---

## Import from HTML Files

HTML is one of the formats that can be imported as an XML document. The steps needed are:

1. Go to menu **File > Import > HTML File ...**  
The **Import HTML** dialog box is displayed.
2. Enter the URL of the HTML document.
3. Select the type of the result XHTML document:
  - XHTML 1.0 Transitional
  - XHTML 1.0 Strict
4. Click the **OK** button.

The resulting document is an XHTML file containing a DOCTYPE declaration that references the XHTML DTD definition on the Web. The parsed content of the imported file is transformed to XHTML Transitional or XHTML Strict depending on what radio button you chose when performing the import operation.

---

## Import from Text Files

The steps for importing a text file into an XML file are the following:

1. Go to menu **File > Import > Text File...**  
The **Select text file** dialog box is displayed.
2. Select the URL of the text file.
3. Select the encoding of the text file.
4. Click the **OK** button.

The **Import Criteria** dialog box is displayed:

The input data is displayed in a tabular form. The **XML Import Preview** panel contains an example of what the generated XML document looks like. The names of the XML elements and the transformation of the first five lines from the text file are displayed in the **Import settings** section. All data items in the input are converted by default to element content, but this can be overridden by clicking the individual column headers. Clicking once a column header causes the data from this column to be used as attribute values of the row elements. Click the second time and the column's data is ignored when generating the XML file. You can cycle through these three options by continuing to click the column header. The following symbols decorate the column header to indicate the type of content that column is converted to:

- <> symbols for data columns converted to element content

- = symbol for data columns converted to attribute content
- x symbol for ignored data

**5.** Select the field delimiter for the import settings:

- Comma;
- Semicolon;
- Tab;
- Space;
- Pipe.

**6.** Set other optional settings of the conversion.

The dialog box offers the following settings:

- **First row contains field names** - If the option is enabled, you will notice that the table has moved up. The default column headers are replaced (where such information is available) by the content of the first row. In other words, the first row is interpreted as containing the field names. The changes are also visible in the preview of the XML document. To return to default settings (where the first row is interpreted as containing data and not fields names), simply uncheck the option.
- **Change labels** - This button opens a new dialog box that allows you to edit the names of the root and row elements, change the XML name and the conversion criterion.

The XML names can be edited by double-clicking the desired item and entering the required label. The conversion criterion can also be modified by selecting one of the drop-down list options: **ELEMENT**, **ATTRIBUTE**, or **SKIPPED**.

- **Open in editor** - If checked, the new XML document created from the imported text file is opened in the editor.
- **Save in file** - If checked, the new XML document is saved at the specified path.

 **Note:** If only **Open in editor** is checked, the newly created document is opened in the editor, but as an unsaved file.

## Import Content Dynamically

Along with the built-in support for various useful URL protocols (such as HTTP or FTP), Oxygen XML Editor also provides special support for a *convert* protocol that can be used to chain predefined processors to import content from various sources dynamically.

A dynamic conversion URL chains various processors that can be applied in sequence on a target resource and has the following general syntax:

```
convert:/processor=xslt;ss=urn:processors:excel2d.xsl/processor=excel!/urn:files:sample.xls
```

The previous example first applies a processor called `excel` on a target identified by the identifier `urn:files:sample.xls` and converts the Excel™ resource to XML. The second applied processor (`xslt`) applies an XSLT stylesheet identified using the identifier `urn:processors:excel2d.xsl` over the content resulting from the first applied processor. These identifiers are all mapped to real resources on disk via an *XML catalog* that is configured in the application, as in the following example:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
 <rewriteURI uriStartString="urn:files:" rewritePrefix=".//resources//"/>
 <rewriteURI uriStartString="urn:processors:" rewritePrefix=".//processors//"/>
</catalog>
```

This type of URL can be opened in the application by using the **Open URL...** action from the **File** menu. It can also be referenced from existing XML resources via `xi:include` or from *DITA Maps* as topic references.

A GitHub project that contains various dynamic conversion samples for producing DITA content from various sources (and then publishing it) can be found here: <https://github.com/oxygenxml/dita-glass>.

## Conversion Processors

A set of predefined conversion processors is provided in Oxygen XML Editor out-of-the box. Each processor has its own parameters that can be set to control the behavior of the conversion process. All parameters that are resolved to resources are passed through the XML catalog mapping.

The following predefined conversion processors are included:

- **xslt Processor** - Converts an XML input using XSLT 2.0 processing. The `ss` parameter indicates the stylesheet resource to be loaded. All other specified parameters will be set as parameters to the XSLT transformation.

```
convert:/processor=xslt;ss=urn:processors:convert.xsl;p1=v1!;/urn:files:sample.xml
```

- **xquery Processor** - Converts an XML input using XQuery processing. The `ss` parameter indicates the XQuery script to be loaded. All other specified parameters will be set as parameters to the XSLT transformation.

```
convert:/processor=xquery;ss=urn:processors:convert.xquery;p1=v1!;/urn:files:sample.xml
```

- **excel Processor** - Converts an Excel™ input to an XML format that can be later converted by other piped processors. It has a single parameter `sn`, which indicates the name of the sheet that needs to be converted. If this parameter is missing, the XML will contain the combined content of all sheets included in the Excel™ document.

```
convert:/processor=excel;sn=test!;/urn:files:sample.xls
```

- **java Processor** - Converts an input to another format by applying a specific Java method. The `jars` parameter is a comma separated list of JAR libraries or folders, from which libraries will be loaded. The `ccn` parameter is the fully qualified name of the conversion class that will be instantiated. The conversion class needs to have a method with the following signature:

```
public void convert(String systemID, String originalSourceSystemID, InputStream is, OutputStream os, LinkedHashMap<String, String> properties) throws IOException
```

```
convert:/processor=java;jars=libs;ccn=test.JavaToXML! /urn:files:java/WSEditorBase.java
```

- **js Processor** - Converts an input to another format by applying a JavaScript method. The `js` parameter indicates the script that will be used. The `fn` parameter is the name of the method that will be called from the script. The method must take a string as an argument and return a string. If any of the parameters are missing, an error is thrown and the conversion stops.

```
convert:/processor=js;js=urn:processors:md.js;fn=convertExternal!;/urn:files:sample.md
```

- **json Processor** - Converts a JSON input to XML. It has no parameters.

```
convert:/processor=json!;/urn:files:personal.json
```

- **xhtml Processor** - Converts HTML content to well-formed XHTML. It has no parameters.

```
convert:/processor=xhtml!;/urn:files:test.html
```

- **wrap Processor** - Wraps content in a tag name making it well-formed XML. The `rn` parameter indicates the name of the root tag to use. By default, it is `wrapper`. The `encoding` parameter specifies the encoding that should be used to read the content. By default, it is `UTF8`. As an example, this processor can be used if you want to process a comma-separated values file with an XSLT stylesheet to produce XML content. The CSV file is first wrapped as well-formed XML, which is then processed with an `xslt` processor.

```
convert:/processor=wrap!;/urn:files:test.csv
```

## Reverse Conversion Processors

All processors defined above can also be used for saving content back to the target resource if they are defined in the URL as reverse processors. Reverse processors are evaluated right to left. These reverse processors allow *round-tripping* content to and from the target resource.

As an example, the following URL converts HTML to DITA when the URL is opened using the h2d.xsl stylesheet and converts DITA to HTML when the content is saved in the application using the d2h.xsl stylesheet.

```
convert:/processor=xslt;ss=h2d.xsl/rprocessor=xslt;ss=d2h.xsl!/urn:files:sample.html
```



---

# Chapter

# 18

---

## Content Management System (CMS) Integration

---

**Topics:**

- *Integration with Documentum CMS (deprecated)*
- *Integration with Microsoft SharePoint*

This chapter explains how you can integrate Oxygen XML Editor with a content management system (CMS), to edit the data stored in the CMS directly in Oxygen XML Editor. Oxygen XML Editor offers support for Documentum CMS and Microsoft SharePoint, but other CMSs can use the *plugin support* for similar integrations.

## Integration with Documentum (CMS) (deprecated)

---

 **Important:** Starting with version 17.0, the support for Documentum (CMS) is deprecated and will no longer be actively maintained.

Oxygen XML Editor provides support for browsing and managing Documentum repositories in the Data Source Explorer. You can easily create new resources on the repository, copy and move them using contextual actions or the drag and drop support, edit and transform the documents in the editor. The operations that can be performed on repository resources are described in the [Documentum \(CMS\) actions](#) section.

Oxygen XML Editor supports Documentum (CMS) version 6.5 and 6.6 with *Documentum Foundation Services 6.5 or 6.6* installed.

 **Attention:**

It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

### Configure Connection to Documentum Server

This section explains how to configure a connection to a Documentum server.

#### How to Configure a Documentum (CMS) Data Source

Available in the Enterprise edition only.

To configure a Documentum (CMS) data source you need the Documentum Foundation Services Software Development Kit (*DFS SDK*) corresponding to your server version. The *DFS SDK* can be found in the Documentum (CMS) server installation kit or it can be downloaded from [EMC Community Network](#).

 **Note:** The *DFS SDK* can be found in the form of an archive named, for example, *emc-dfs-sdk-6.5.zip* for Documentum (CMS) 6.5.

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. In the **Data Sources** panel click the **New** button.
3. Enter a unique name for the data source.
4. Select **Documentum (CMS)** from the driver type combo box.
5. Press the **Choose DFS SDK Folder** button.
6. Select the folder where you have unpacked the *DFS SDK* archive file.

If you have indicated the correct folder the following Java libraries (jar files) will be added to the list (some variation of the library names is possible in future versions of the *DFS SDK*):

- lib/java/emc-bpm-services-remote.jar
- lib/java/emc-ci-services-remote.jar
- lib/java/emc-collaboration-services-remote.jar
- lib/java/emc-dfs-rt-remote.jar
- lib/java/emc-dfs-services-remote.jar
- lib/java/emc-dfs-tools.jar
- lib/java/emc-search-services-remote.jar
- lib/java/ucf/client/ucf-installer.jar
- lib/java/commons/\*.jar (multiple jar files)
- lib/java/jaxws/\*.jar (multiple jar files)
- lib/java/utils/\*.jar (multiple jar files)

 **Note:** If for some reason the jar files are not found, you can add them manually by using the **Add Files** and **Add Recursively** buttons and navigating to the lib/java folder from the *DFS SDK*.

7. Click the **OK** button to finish the data source configuration.

## How to Configure a Documentum (CMS) Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a Documentum (CMS) server are the following:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the previously configured Documentum (CMS) data sources in the **Data Source** combo box.
5. Fill-in the connection details:
  - **URL** - The URL to the Documentum (CMS) server: `http://<hostname>:<port>`
  - **User** - The user name to access the Documentum (CMS) repository.
  - **Password** - The password to access the Documentum (CMS) repository.
  - **Repository** - The name of the repository to log into.
6. Click the **OK** button to finish the configuration of the connection.

## Known Issues

The following are known issues with the Documentum (CMS):

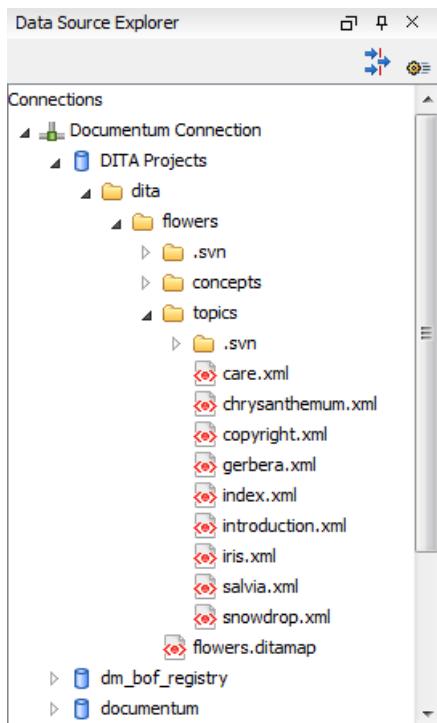
1. Please note that there is a known problem in the UCF Client implementation for Mac OS X from Documentum 6.5 which prevents you from viewing or editing XML documents from the repository on Mac OS X. The UCF Client is the component responsible for file transfer between the repository and the local machine. This component is deployed automatically from the server. Documentum 6.6 does not exhibit this problem.
2.  **Note:** This issue was reproduced with Documentum 6.5 SP1. In Documentum 6.6 this is no longer reproducing.
2. In order for the Documentum driver to work faster on Linux, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen XML Editor startup scripts (or in the `*.vmoptions` file) the system property:

```
-Djava.security.egd=file:/dev/.urandom
```

## Documentum (CMS) Actions in the Data Source Explorer View

Oxygen XML Editor allows you to browse the structure of a Documentum repository in the **Data Source Explorer** view and perform various operations on the repository resources.

You can drag and drop folders and resources to other folders to perform move or copy operations with ease. If the drag and drop is between resources (drag the child item to the parent item) you can create a relationship between the respective resources.



**Figure 348: Browsing a Documentum repository**

### Actions Available on Connection

The contextual menu of a Documentum (CMS) connection in the **Data Source Explorer** view offers the following actions:

#### ⚙ Configure Database Sources

Opens the *Data Sources preferences page* where you can configure both data sources and connections.

#### New Cabinet

Creates a new cabinet in the repository. The cabinet properties are:

- **Type** - The type of the new cabinet (default is **dm\_cabinet**).
- **Name** - The name of the new cabinet.
- **Title** - The title property of the cabinet.
- **Subject** - The subject property of the cabinet.

#### ⟳ Refresh

Refreshes the connection.

### Actions Available on Cabinets / Folders

The actions available on a Documentum (CMS) cabinet in the **Data Source Explorer** view are the following:

#### 📁 New Folder

Creates a new folder in the current cabinet / folder. The folder properties are the following:

- **Path** - Shows the path where the new folder will be created.
- **Type** - The type of the new folder (default is **dm\_folder**).
- **Name** - The name of the new folder.
- **Title** - The title property of the folder.
- **Subject** - The subject property of the folder.

## New Document

Creates a new document in the current cabinet / folder. The document properties are the following:

- **Path** - Shows the path where the new document will be created.
- **Name** - The name of the new document.
- **Type** - The type of the new document (default is **dm\_document**).
- **Format** - The document content type format.

## Import

Imports local files / folders in the selected cabinet / folder of the repository. Actions available when performing an import:

- **Add Files** - Opens a file browse dialog box and allows you to select files to add to the list.
- **Add Folders** - Opens a folder browse dialog box that allows you to select folders to add to the list. The subfolders will be added recursively.
- **Edit** - Opens a dialog box where you can change the properties of the selected file / folder from the list.
- **Remove** - Removes the selected files / folders from the list.

## Rename

Changes the name of the selected cabinet / folder.

## Copy

Copies the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop while holding the (**Ctrl (Meta on Mac OS)**) key pressed.

## Move

Moves the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop.

## Delete

Deletes the selected cabinet / folder from the repository. The following options are available:

- **Folder(s)** - Allows you to delete only the selected folder or to delete recursively the folder and all subfolders and objects.
- **Version(s)** - Allows you to specify what versions of the resources will be deleted.
- **Virtual document(s)** - Here you can specify what happens when virtual documents are encountered. They can be either deleted either by themselves or together with their descendants.

## Refresh

Performs a refresh of the selected node's sub-tree.

## Properties

Displays the list of properties of the selected cabinet / folder.

## Actions Available on Resources

The actions available on a Documentum (CMS) resource in the **Data Source Explorer** view are the following:

## Edit

Checks out (if not already checked out) and opens the selected resource in the editor.

## Edit with

Checks out (if not already checked out) and opens the selected resource in the specified editor / tool.

## Open (Read-only)

Opens the selected resource in the editor. The resources are marked as read-only in the editor using a lock icon on the file tab. If you want to edit those resources, enable the *Can edit read only files* option.

## Open with

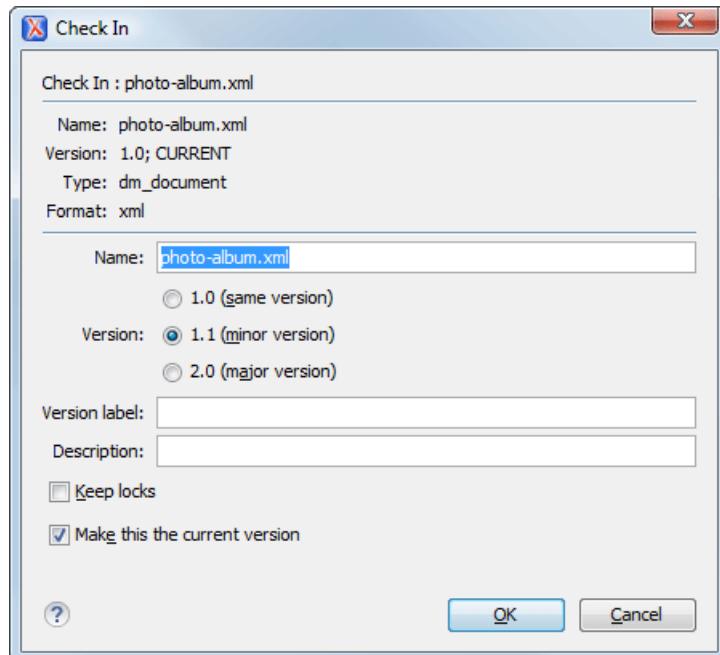
Opens the selected resource in the specified editor / tool.

### Check Out

Checks out the selected resource from the repository. The action is not available if the resource is already checked out.

### Check In

Checks in the selected resource (commits changes) into the repository. The action is only available if the resource is checked out.



**Figure 349: Check In Dialog Box**

The following resource properties are available:

- **Name** - The resource name in the repository.
- **Version** - Allows you to choose what version the resource will have after being checked in.
- **Version label** - The label of the updated version.
- **Description** - An optional description of the resource.
- **Keep Locks** - When this option is enabled, the updated resource is checked into the repository but it also keeps it locked.
- **Make this the current version** - Makes the updated resource the current version (will have the *CURRENT* version label).

### Cancel Checkout

Cancels the checkout process and loses all modifications since the checkout. Action is only available if the resource is checked out.

### Export

Allows you to export the resource and save it locally.

### Rename

Changes the name of the selected resource.

### Copy

Copies the selected resource in a different location in the tree. Action is not available on virtual document descendants. This action can also be performed with drag and drop while holding the **Ctrl (Meta on OS X)** key pressed.

### Move

Moves the selected resource in a different location in the tree. Action is not available on virtual document descendants and on checked out resources. This action can also be performed with drag and drop.

 **Delete**

Deletes the selected resource from the repository. Action is not available on virtual document descendants and on checked out resources.

**Add Relationship**

Adds a new relationship for the selected resource. This action can also be performed with drag and drop between resources.

**Convert to Virtual Document**

Allows you to convert a simple document to a virtual document. Action is available only if the resource is a simple document.

**Convert to Simple Document**

Allows you to convert a virtual document to a simple document. Action is available only if the resource is a virtual document with no descendants.

**Copy location**

Allows you to copy to clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.

 **Refresh**

Performs a refresh of the selected resource.

 **Properties**

Displays the list of properties of the selected resource.

## Transformations on DITA Content from Documentum (CMS)

Oxygen XML Editor comes with the DITA Open Toolkit which is able to transform a DITA map to various output formats. However DITA Open Toolkit requires local DITA files so first you need to check out a local version of your DITA content. Once you have a local version of a DITA map just load it in *the DITA Maps Manager view* and run one of the DITA transformations that are predefined in Oxygen XML Editor or a customization of such a predefined DITA transformation.

## Integration with Microsoft SharePoint

---

This section explains how to work with a SharePoint connection in the **Data Source Explorer** view.



**Note:** The SharePoint connection is available in the Enterprise edition.



**Note:** You can access documents stored on SharePoint Online for Office 365.

To watch our video demonstration about connecting to a repository located on a SharePoint server and using SharePoint, go to [http://www.oxygenxml.com/demo/SharePoint\\_Support.html](http://www.oxygenxml.com/demo/SharePoint_Support.html).and SharePoint Online for Office 365

## How to Configure a SharePoint Connection

By default Oxygen XML Editor contains a predefined **SharePoint** data source. Use this data source to create a connection to a SharePoint server which will be available in *the Data Source Explorer view*.

Follow these steps to configure a SharePoint connection:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select SharePoint in the **Data Source** combo box.
5. Fill-in the connection details:
  - a) Set the URL to the SharePoint repository in the field **SharePoint URL**.

- b) Set the server domain in the **Domain** field.
- c) Set the user name to access the SharePoint repository in the **User** field.
- d) Set the password to access the SharePoint repository in the **Password** field.

To watch our video demonstration about connecting to repository located on a SharePoint server, go to [http://www.oxygenxml.com/demo/SharePoint\\_Support.html](http://www.oxygenxml.com/demo/SharePoint_Support.html).

## The SharePoint Browser View

To display the **SharePoint Browser** view, go to **Window > Show View > SharePoint Browser**. This view allows you to connect to a SharePoint repository and perform SharePoint-specific actions on the available resources.

The screenshot shows the SharePoint Browser window. At the top, there's a header bar with a 'Site' dropdown set to 'MySharePointSite' and some icons for disconnecting and settings. Below the header is a tree view of the SharePoint site structure under '<oxygen/> Team Site'. The tree includes 'Automatic Tests', 'Test Samples', 'Documents [All Documents]', 'MicroFeed', 'Site Assets', 'Site Pages', 'Oxygen XML Author Applet', 'QA Test Site', 'Documents [Only DITA]', 'Form Templates', 'Site Assets', 'Site Pages [All Pages]', 'SitePages', and 'Style Library'. To the right of the tree view is a table listing files. The columns are 'Type', 'Name', 'Modified', 'ID', and 'Version'. The table contains 45 rows of file information, such as 'autumnFlowers.dita' modified on 2014-01-27 at 04:53:57, ID 13, Version 1.0, and 'lilac.dita' modified on 2014-01-27 at 04:55:04, ID 45, Version 1.1.

Type	Name	Modified	ID	Version
	autumnFlowers.dita	2014-01-27 04:53:57	13	1.0
	glossaryBulb.dita	2014-01-27 04:53:57	14	1.0
	glossaryCultivar.dita	2014-01-27 04:53:59	15	1.0
	glossaryGenus.dita	2014-01-27 04:54:02	16	1.0
	glossaryPanicle.dita	2014-01-27 04:54:02	17	1.0
	glossaryPerennial.dita	2014-01-27 04:54:03	18	1.0
	glossaryPollination.dita	2014-01-27 04:54:04	19	1.0
	glossaryRhizome.dita	2014-01-27 04:54:05	20	1.0
	glossarySepal.dita	2014-01-27 04:54:06	21	1.0
	springFlowers.dita	2014-01-27 04:54:07	22	1.0
	summerFlowers.dita	2014-01-27 04:54:09	23	1.0
	winterFlowers.dita	2014-01-27 04:54:10	24	1.0
	gardenPreparation.dita	2014-01-27 04:54:33	35	1.0
	pruning.dita	2014-01-27 04:54:38	36	1.0
	care.dita	2014-01-27 04:54:40	38	1.0
	copyright.dita	2014-01-27 04:54:43	39	1.0
	chrysanthemum.dita	2014-01-27 04:54:50	41	1.0
	gardenia.dita	2014-01-27 04:54:52	42	1.0
	gerbera.dita	2014-01-27 04:54:53	43	1.0
	iris.dita	2014-01-27 04:55:01	44	1.0
	lilac.dita	2014-01-27 04:55:04	45	1.1

**Figure 350: SharePoint Browser View**

The view is split in several functional areas:

### Connection Area

The following controls are available:

- The **Site** combo box allows you to select and connect to an already [defined SharePoint connection](#).
- The **Disconnect** action terminates the current connection.
- The **Settings** drop-down menu contains actions that help you to quickly define a new connection or manage the existing ones from the **Data Source** options page: **New SharePoint Connection** and **Configure Database Sources**. Also, here you can choose one of the predefined view layouts.

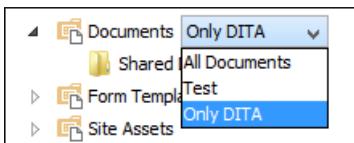
## SharePoint Site Navigation Area

If there is no connection selected in the **Site** combo box, this area is left blank and promotes the actions that allow you to quickly add SharePoint connections. Otherwise, the navigation area presents the SharePoint site structure in a tree-like fashion displaying the following node types: *sites* (🌐), *libraries*, and *folders*.

Depending on a node's type, a contextual menu offers customized actions that can be performed on that node.

-  **Note:** The contextual menu of a folder allows you to create new folders, new documents, and to rename and delete the folder.
-  **Note:** The rename and delete actions are not available for library root folders (folders located at first level in a SharePoint library).

Each library node display next to its name a drop down box where you can select the current library *view*. This functionality is also available on the node's contextual menu, under the **Current View** submenu.



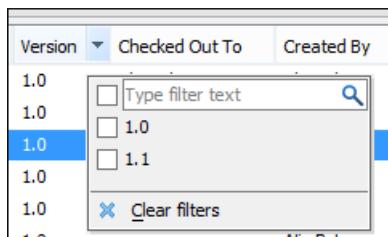
## Folder Content Area

The content of a folder is displayed in a tabular form, where each row represents the properties of a folder or document. The list of columns and the way the documents and folders are organized depends on the currently selected view of the parent library.

Action	Description	Available for	
		folders	documents
 <b>Open</b>	Displays the content of the current selected folder. Opens the current document for editing.		
<b>Rename...</b>	Renames the current node on server.		
 <b>Delete...</b>	Deletes the current node from the server.		
<b>Copy Location</b>	Copies to clipboard the URL of the current node.		
 <b>Check Out</b>	Reserves the current document for your use so that other users cannot change it while you are editing it.		
<b>Check In...</b>	Commits on the server the changes you made to the document, so that other users can see them. It also makes the document available for editing to other users.		
<b>Discard Check Out</b>	Discards the previous checkout operation, making the file available for editing to other users.		
 <b>Refresh</b>	Queries the server in order to refresh the available properties of the current node.		

You can filter and sort the displayed items. To display the available filters of a column, click the filter widget located on the column's header. You can apply multiple filters at the same time.

-  **Note:** A column can be filtered or sorted only if it was configured this way on the server side.



## SharePoint Connection Actions

This section explains the actions that are available on a SharePoint connection in the **Data Source Explorer** view.

### Actions Available at Connection Level

The contextual menu of a SharePoint connection in the **Data Source Explorer** view contains the following actions:

#### Configure Database Sources...

Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

#### Disconnect

Stops the connection.

#### New Folder...

Creates a new folder on the server.

#### Import Files...

Allows you to add a new file on the server.

#### Refresh

Performs a refresh of the connection.

#### Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

### Actions Available at Folder Level

The contextual menu of a folder node in a SharePoint connection in the **Data Source Explorer** view contains the following actions:

#### New File

Creates a new file on the server in the current folder.

#### New Folder...

Creates a new folder on the server.

#### Import Folders...

Imports folders on the server.

#### Import Files

Allows you to add a new file on the server in the current folder.

#### Cut

Removes the current selection and places it in the clipboard.

#### Copy

Copies the current selection.

#### Paste

Pastes the copied selection.

**Rename**

Allows you to change the name of the selected folder.

** Delete**

Removes the selected folder.

** Refresh**

Refreshes the sub-tree of the selected node.

** Find/Replace in Files...**

Allows you to find and replace text in multiple files from the server.

**Actions Available at File Level**

The contextual menu of a file node in a SharePoint connection in the **Data Source Explorer** view contains the following actions:

** Open**

Allows you to open the selected file in the editor.

** Cut**

Removes the current selection and places it in the clipboard.

** Copy**

Copies the current selection into the clipboard.

**Copy Location**

Copies an application specific URL for the selected resource to the clipboard. You can use this URL for various actions like opening or transforming the resources.

**Check Out**

Checks out the selected document on the server.

**Check In**

Checks in the selected document on the server. This action opens the **Check In** dialog. In this dialog, the following options are available:

- **Minor Version** - increments the minor version of the file on the server
- **Major Version** - increments the major version of the file on the server
- **Overwrite** - overwrites the latest version of the file on the server
- **Comment** - allows you to comment on a file that you check in

**Discard Check Out**

Discards the previous checkout operation, making the file available for editing to other users.

**Rename**

Allows you to change the name of the selected file.

** Delete**

Removes the selected file.

** Refresh**

Performs a refresh of the selected node.

** Properties**

Displays the properties of the current file in a **Properties** dialog box.

** Find/Replace in Files...**

Allows you to find and replace text in multiple files from the server.

 **Note:** The **Check In**, **Check Out**, and **Discard Check Out** options are available in the Enterprise edition only.



---

# Chapter

# 19

---

## Tools

---

**Topics:**

- [\*SVN Client\*](#)
- [\*Tree Editor\*](#)
- [\*Comparing and Merging Documents\*](#)
- [\*XML Digital Signatures\*](#)
- [\*Large File Viewer\*](#)
- [\*Hex Viewer\*](#)
- [\*Integrating External Tools\*](#)

Oxygen XML Editor ships with a set of tools oriented to XML related tasks. You have access to a revision control system, a comparing and merging solution and also to other tools like a large file viewer and a hex viewer.

## SVN Client

---

The Syncro SVN Client is a client application for the Apache Subversion™ version control system, compatible with Subversion 1.6, 1.7 and 1.8 servers. It manages files and directories that change over time and are stored in a central repository. The version control repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to access older versions of your files and examine the history of how and when your data changed.

To start Syncro SVN Client, go to **Tools > SVN Client**.

### Main Window

This section explains the main window of Syncro SVN Client.

#### Views

The main window consists of the following views:

- ***Repositories view*** - Allows you to define and manage Apache Subversion™ repository locations.
- ***Working Copy view*** - Allows you to manage with ease the content of the working copy.
- ***History view*** - Displays information (author name, revision number, commit message) about the changes made to a resource during a specified period of time.
- ***Editor view*** - Allows you to edit different types of text files, with full syntax-highlight.
- ***Annotations view*** - Displays a list with information regarding the structure of a document (author and revision for each line of text).
- ***Compare view*** - Displays the differences between two revisions of a text file from the working copy.
- ***Image Preview*** - Allows you to preview standard image files supported by Syncro SVN Client: JPG, GIF and PNG.
- ***Compare Images view*** - Displays two images side by side.
- ***Properties view*** - Displays the SVN properties of a resource under version control.
- ***Console view*** - Displays information about the currently running operation, similar with the output of the Subversion command line client.
- ***Help view*** - Shows information about the currently selected view.

The main window's status bar presents in the left side the operation in progress or the final result of the last performed action. In the right side there is a progress bar for the running operation and a stop button to cancel the operation.

### Main Menu

The main menu of the Syncro SVN Client is composed of the following menus:

- **File** menu:

#### New submenu:

##### **New File...**

This operation creates a new file as a child of the selected folder from the ***Repositories view*** tree or the ***Working Copy view*** tree, depending on the view that was last used. Note that for the ***Working Copy view***, the file is added to *version control* only if the selected folder is under *version control*.

##### **New Folder... (Ctrl (Command on OS X) + Shift + F)**

This operation creates a new folder as a child of the selected folder from the ***Repositories view*** tree or the ***Working Copy view*** tree, depending on the view that was last used. Note that for the ***Working Copy view***, the file is added to *version control* only if the selected folder is under *version control*.

##### **New External Folder... (Ctrl (Command on OS X) + Shift + W)**

This operation allows you to add a new external definition on the selected folder. An external definition is a mapping of a local directory to a ***URL of a versioned directory***, and ideally a particular revision, stored in the **svn:externals** property of the selected folder.

 **Tip:** You can specify a particular revision of the external item by using a *peg revision* at the end of the URL (for example, URL@rev1234). You can also use peg revisions to access external items that were deleted, moved, or replaced.

The URL used in the external definition format can be relative. You can specify the repository URL that the external folder points to by using one of the following relative formats:

- `./` - Relative to the URL of the directory on which the `svn:externals` property is set.
- `^/` - Relative to the root of the repository in which the `svn:externals` property is versioned.
- `//` - Relative to the scheme of the URL of the directory on which the `svn:externals` property is set.
- `/` - Relative to the root URL of the server in which the `svn:externals` property is versioned.

 **Important:** To change the target URL of an external definition, or to delete an external item, do the following:

1. Modify or delete the item definition found in the `svn:externals` property that is set on the parent folder.
2. For the change to take effect, use the **Update** operation on the parent folder of the external item.

 **Note:** Syncro SVN Client does not support definitions of local relative external items.

#### **Open (Ctrl (Command on OS X) + O)**

This action opens the selected file in an editor where you can modify it. The action is active only when a single item is selected. The action opens a file with the internal editor or the external application associated with that file type. This action works on any file selection from the **Repositories view**, **Working Copy view**, **History view**, or **Directory Change Set view**, depending on the view that was last used to invoke it. In the case of a folder, the action opens the selected folder with the system application for folders (for example, Windows Explorer on Windows or Finder on OS X, etc). Note that opening folders is available only for folders selected in the **Working Copy view**.

#### **Open with... (Ctrl (Command on OS X) + Shift + O)**

Displays the **Open with** dialog for specifying the editor in which the selected file is opened. If multiple files are selected only external applications can be used to open the files. This action works on any file selection from **Repositories view**, **Working Copy view**, **History view**, or **Directory Change Set view**, depending on the view that was last used to invoke it.

#### **Show in Explorer/Show in Finder**

Opens the parent directory of the selected working copy file and selects the file.

#### **Save (Ctrl (Command on OS X) + S)**

Saves the local file currently opened in the editor or the **Compare** view.

#### **Save as...**

Saves any file selected in the **Repositories**, **History**, or **Directory Change Set** view.

#### **Copy URL Location (Ctrl (Command on OS X) + Alt + U)**

Copies the URL location of the resource currently selected in the **Repositories** view to clipboard.

#### **Copy to...**

Copies the currently selected resource, either in **Repositories** or **Working copy** view, to a specified location.

 **Note:** This action can also be used from **History** and **Directory Change Set** views to recover older versions of a repository item.

#### **Move to... (Ctrl (Command on OS X) + M)**

Moves the currently selected resource, either in **Repositories** or **Working copy** view, to a specified location.

#### **Rename... (F2)**

Renames the resource currently selected, either in **Repositories** or **Working copy** view.

### **Delete (Delete)**

Deletes the resource currently selected either, in **Repositories** or **Working copy** view.

#### **Locking:**

-  **Scan for locks... (Ctrl (Command on OS X) + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. This is only active for resources under *version control*. For more details see [Scanning for locks](#).
-  **Lock... (Ctrl (Command on OS X) + K)** - Allows you to lock certain files that need exclusive access. You can write a comment describing the reason for the lock and you can also force (*steal*) the lock. This action is active only on files under *version control*. For more details on the use of this action see [Locking a file](#).
-  **Unlock... (Ctrl (Command on OS X) + Alt + K)** - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

### **Show SVN Properties (Ctrl (Command on OS X) + P)**

Opens the [Properties view](#) and displays the SVN properties for a selected resource from [Repositories view](#) or [Working Copy view](#), depending on the view that was last used to invoke it.

### **Show SVN Information (Ctrl (Command on OS X) + I)**

Provides additional information for a selected resource. For more details, go to [Obtain information for a resource](#).

### **Exit (Ctrl (Command on OS X) + Q)**

Closes the application.

- **Edit menu:**

### **Undo (Ctrl (Command on OS X) + Z)**

Undo edit changes in the local file that is currently opened in the editor or the **Compare** view.

### **Redo (Ctrl (Command on OS X) + Y)**

Redo edit changes in the local file that is currently opened in the editor or the **Compare** view.

### **Cut (Ctrl (Command on OS X) + X)**

Cut selection from the local file that is currently opened in the editor view or the **Compare** view to clipboard.

### **Copy (Ctrl (Command on OS X) + C)**

Copy selection from the local file that is currently opened in the editor or the **Compare** view to clipboard.

### **Paste (Ctrl (Command on OS X) + V)**

Paste selection from clipboard into the local file that is currently opened in editor or the **Compare** view.

### **Find/Replace (Ctrl (Command on OS X) + F)**

Perform find and replace operations in the local file that is currently opened in the editor or the **Compare** view.

### **Find Next (F3)**

Go to the next match using the same find options of the last find operation. This action runs in the editor panel and in any non-editable text area (for example, the **Console** view).

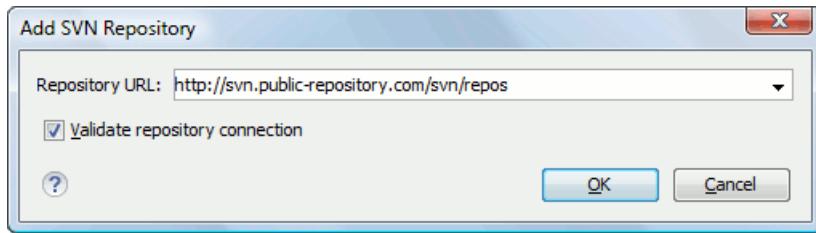
### **Find Previous (Shift + F3)**

Go to the previous match using the same find options of the last find operation. This action runs in the editor panel and in any non-editable text area (for example, the **Console** view).

- **Repository menu:**

### **New Repository Location... (Ctrl Alt N (Command Alt N on OS X))**

Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.



**Figure 351: Add SVN Repository Dialog Box**

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

**Edit Repository Location...** (**Ctrl Alt E (Command Alt E on OS X)**)

Context-dependent action that allows you to edit the selected repository location using the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.

**Change the Revision to Browse...** (**Ctrl Alt Shift B (Command Alt Shift B on OS X)**)

Context-dependent action that allows you to change the selected repository revision using the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.

**Remove Repository Location...** (**Ctrl Alt Shift R (Command Alt Shift R on OS X)**)

Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.

**Refresh (F5)**

Refreshes the resource selected in the **Repositories** view.

**Check out...** (**Ctrl Alt Shift C (Command Alt Shift C on OS X)**)

Allows you to create a working copy from a repository directory, on your local file system. To read more about this operation, see the section [Check out a working copy](#).

**Export...**

Opens [the Export dialog box](#) that allows you to configure options for exporting a folder from the repository to the local file system.

**Import:**

**Import folder...** (**Ctrl Alt Shift M (Command Alt Shift M on OS X)**)

Allows you to import the contents of a specified folder from the file system into the selected folder in a repository. To read more about this operation, see the section [Importing resources into a repository](#).

**Note:** The difference between the **Import folder...** and **Share project...** actions is that the latter also converts the selected directory into a working copy.

**Import Files...** (**Ctrl Alt I (Command Alt I on OS X)**)

Imports the files selected from the files system into the selected folder in the repository.

- **Working Copy** menu:

**(on OS X)Working Copies Manager**

Opens a dialog with a list of working copies that the Apache Subversion™ client is aware of. In this dialog you can add existing working copies or remove those that are no longer needed.

**Switch to**

Selects one of the following view modes: **All Files**, **Modified**, **Incoming**, **Outgoing**, or **Conflicts**.

**Refresh (F5)**

Refreshes the state of the selected resources or of the entire working copy (if there is no selection).

 **Synchronize (Ctrl (Command on OS X) + Shift + S)**

Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.

**Update (Ctrl (Command on OS X) + U)**

Updates all the selected resources that have incoming changes to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.

**Update to revision/depth...**

Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the [sparse checkouts](#) section.

**Commit...**

Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what resources to commit. A directory will always be committed recursively. Unversioned resources will be deselected by default. In the **Commit** dialog you can also enter a comment before sending your changes to the repository.

 **Update all... (Ctrl (Command on OS X) + Shift + U)**

Updates all resources from the working copy that have incoming changes. It performs a recursive update on the synchronized resources.

 **Commit all...**

Commits all the resources with outgoing changes. It is disabled when **Incoming** mode is selected or the synchronization result does not contain resources with outgoing changes. It performs a recursive commit on the synchronized resources.

 **Revert... (Ctrl (Command on OS X) + Shift + V)**

Undoes all local changes for the selected resources. It does not contact the repository and the files are obtained from Apache Subversion™ pristine copy. It is enabled only for modified resources. See [Revert your changes](#) for more information.

**Edit conflict... (Ctrl (Command on OS X) + E)**

Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see [Edit conflicts](#).

 **Mark Resolved (Ctrl (Command on OS X) + Shift + R)**

Instructs the Subversion system that you resolved a conflicting resource. For more information, see [Merge conflicts](#).

 **Mark as Merged (Ctrl (Command on OS X) + Shift + M)**

Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the [Merge conflicts](#) section for more information about how you can solve the pseudo-conflicts.

**Override and Update...**

Drops any outgoing change and replaces the local resource with the HEAD revision. This action is available on resources with outgoing changes, including conflicting ones. See the [Revert your changes](#) section.

**Override and Commit...**

Drops any incoming changes and sends your local version of the resource to the repository. This action is available on conflicting resources. For more information see [Drop incoming modifications](#).

**Mark as copied**

You can use this action to mark an item from the working copy as a copy of another item under *version control*, when the copy operation was performed outside of an SVN client. The **Mark as copied** action is available when you select two items (both the new item and source item), and it depends on the state of the source item.

**Mark as moved**

You can use this action to mark an item from the working copy as being moved from another location of the working copy, when the move operation was performed outside of an SVN client. The **Mark as moved** action is available

when you select two items from different locations (both the new item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

#### **Mark as renamed**

You can use this action to mark an item from the working copy as being renamed outside of an SVN client. The **Mark as renamed** action is available when you select two items from the same directory (both the new item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

#### **Add to "svn:ignore"...** (**Ctrl (Command on OS X) + Alt + I**)

Allows you to add files that should not participate in the *version control* operations inside your working copy . This action can only be performed on resources not under *version control*. It actually modifies the value of the `svn:ignore` property in the parent directory of the resource. Read more about this in the *Ignore Resources Not Under Version Control* section.

#### **Add to version control...** (**Ctrl (Command on OS X) + Alt + V**)

Allows you to add resources that are not under *version control*. For further details, see *Add Resources to Version Control* section.

#### **Remove from version control**

Schedules selected items for deletion from repository upon the next commit. The items are not removed from the file system after committing.

#### **Clean up** (**Ctrl (Command on OS X) + Shift + C**)

Performs a maintenance cleanup operation on the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. This is useful when you already know where the problem originated and want to fix it as quickly as possible. It is only active for resources under *version control*.

#### **Expand all** (**Ctrl (Command on OS X) + Alt + X**)

Displays all descendants of the selected folder. The same behavior is obtained by double-clicking on a collapsed folder.

#### **Collapse all** (**Ctrl (Command on OS X) + Alt + Z**)

Collapses all descendants of the selected folder. The same behavior is obtained by double-clicking on a expanded folder.

- **Compare** menu:

#### **Perform Files Differencing**

Performs a comparison between the source file and target file.

#### **Next Block of Changes** (**Ctrl . (Command . on OS X)**)

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes.

 **Note:** A change block groups one or more consecutive lines that contain at least one change.

#### **Previous Block of Changes** (**Ctrl , (Command , on OS X)**)

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes.

#### **Next Change** (**Ctrl Shift . (Command Shift . on OS X)**)

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change or when there are no changes.

#### **Previous Change** (**Ctrl Shift , (Command Shift , on OS X)**)

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change or when there are no changes.

 **Last Change (Ctrl E (Command E on OS X))**

Jumps to the last change.

 **First Change (Ctrl B (Command B on OS X))**

Jumps to the first change.

 **Copy All Non-Conflicting Changes from Right to Left**

This action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.

 **Copy Change from Right to Left**

This action copies the selected change from the right editor to the left editor.

 **Show Word Level Details**

Provides a word-level comparison of the selected change.

 **Show Character Level Details**

Provides a character-level comparison of the selected change.

 **Format and Indent Both Files (Ctrl Shift P (Command Shift P on OS X))**

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

 **Ignore Whitespaces**

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before the strings are compared they are first normalized and then the whitespace at the beginning and the end of the strings is trimmed.

- **History menu:**

 **Show History... (Ctrl (Command on OS X) + H)**

Displays the history for a SVN resource at a given revision. The resource can be one selected from the **Repositories** view, **Working Copy** view, or from the **Affected Paths** table from the **History** view, depending on which view was last focused when this action was invoked.

 **Show Annotation... (Ctrl Shift A (Command Shift A on OS X))**

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the Annotations view*, along with the history of the file in the **History** view.

**Repositories**

This operation is available for any resource selected from view, **Working Copy** view, **History** view or **Directory Change Sets** view, depending on which view was last focused when this action was invoked.

 **Revision Graph (Ctrl (Command on OS X) + G)**

This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see the section [Revision Graph](#). This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

- **Tools menu:**

**Share project...**

Allows you to [share a new project](#) using an SVN repository. The local project is automatically converted into an SVN working copy.

**Branch / Tag...**

Allows you to copy the selected resource from the **Repositories** view or **Working Copy** view to a branch or tag into the repository. To read more about this operation, see the section [Creating a Branch / Tag](#).

 **Merge... (Ctrl (Command on OS X) + J)**

Allows you to merge the changes made on one branch back into the trunk, or vice versa, using the selected resource from the working copy. To read more about this operation, see the section [Merging](#).

**Switch... (Ctrl (Command on OS X) + Alt + W)**

Allows you to change the repository location of a working copy, or only of a versioned item of the working copy, within the same repository. It is available when the selected item of the working copy is a versioned resource, except for *external* items. To read more about this action, see the [Switching the Repository Location](#) section.

**Relocate...**

Allows you to change the base URL of the root folder of the working copy to a new URL when the base URL of the repository changed. For example, if the repository itself was moved to a different server. This operation is only available for the root item of the working copy. To read more about this operation, see the [Relocate a Working Copy](#) section.

 **Create patch... (Ctrl (Command on OS X) + Alt + P)**

Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see [the section about patches](#).

**Working copy format**

This submenu contains the following two operations:

 **Upgrade**

Upgrades the format of the currently loaded working copy to the newest one known by Syncro SVN Client. This allows you to benefit of all the new features of the client.

 **Downgrade**

Downgrades the format of the currently loaded working copy to SVN 1.7 format. This is useful in case you wish to use older SVN clients with the current working copy, or, by mistake, you have upgraded the format of an older working copy to SVN 1.8.

 **Note:** SVN 1.7 working copies cannot be downgraded to older formats.

See the section [Working Copy Format](#) to read more about this subject.

- **Options menu:**

**Preferences**

Opens the **Preferences** dialog.

**Menu Shortcut Keys...**

Opens the **Preferences** dialog directly on the **Menu Shortcut Keys** option page, where users can configure in one place the keyboard shortcuts available for menu items available in Syncro SVN Client.

**Global Run-Time Configuration**

Allows you to configure SVN general options, that should be used by all the SVN clients you may use:

- **Edit 'config' file** - In this file you can configure various SVN client-side behaviors.
- **Edit 'servers' file** - In this file you can configure various server-specific protocol parameters, including HTTP proxy information and HTTP timeout settings.

**Export Options...**

Allows you to export the current options to an XML file.

**Import Options...**

Allows you to import options you have previously exported.

**Reset Options...**

Resets all your options to the default ones.

**Reset Authentication**

Resets the Subversion authentication information.

- **Window menu:**

**Show View**

Allows you to select the view you want to bring to front.

## Show Toolbar

Allows you to select the toolbar you want to be visible.

## Enable flexible layout

Toggles between a fixed and a flexible layout. When the flexible layout is enabled, you can move and dock the internal views to adapt the application to different viewing conditions and personal requirements.

## Reset Layout

Resets all the views to their default position.

- **Help** menu:

### Help (F1)

Opens the **Help** dialog.

### Use online help (Enabled by default)

If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Oxygen XML Editor attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

### Show Dynamic Help view

Shows the **Dynamic Help** view.

### Report Problem...

Opens a dialog that allows the user to write the description of a problem that was encountered while using the application.

### Support Center

Opens the Support Center web page in a browser.

### About

Opens the **About** dialog.

## Main Toolbar

The toolbar of the Syncro SVN Client SVN Repositories window contains the following actions:



### Check out

Checks out a working copy from a repository. The repository URL and the working copy format must be specified.



### Synchronize

Synchronizes the current working copy with the repository.



### Update All

Updates all resources of the working copy that have an older revision than repository.



### Commit All

Commits all resources of working copy that have a newer version compared to that of the repository.



### Refresh

Refreshes the whole content of the current working copy from disk starting from the root folder. At the end of the operation, the modified files and folders that were not committed to repository yet, are displayed in the **Working Copy** view.



### Compare

The selected resource is compared with:

- the **BASE** revision, when the selected resource is:
  - locally modified and the **All Files** view mode is currently selected (no matter if there are incoming changes)
  - locally modified and there are no incoming changes when any other view mode is selected
- the remote version of the same resource, when remote information is available after a **Synchronize** operation (only when one of **Modified**, **Incoming**, **Outgoing** and **Conflicts** view modes is selected)
- the working copy revision, when the selected resource is from the **History** view



### Show History

Displays the history of the selected resource (from the **Working Copy** or **Repository** views) in the **History** view.



### Show Annotation

Displays the annotations of the selected resource. The selected resource can be in the **Working Copy** or the **History** views.



### Revision Graph

Displays the revision graph of the selected resource. The selected resource can be in the **Working Copy** or the **Repositories** views.

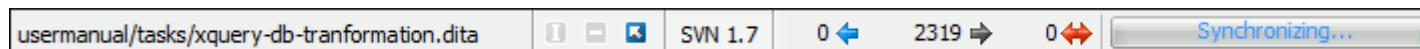


### Enable/Disable flexible layout

Toggles between a fixed and a flexible layout. When the flexible layout is enabled, you can move and dock the internal views to adapt the application to different viewing conditions and personal requirements.

## Status Bar

The status bar of the Syncro SVN Client window displays important details of the current status of the application. This information is available only in the **Working Copy** view.



**Figure 352: Status bar**

The status bar is composed of the following areas:

- the path of the currently processed file from the current working copy (during an operation like **Check out** or **Synchronize**) or the result of the last operation.
- the current status of the following working copy options:
  - Show ignored files ( ).
  - Show deleted files ( ).
  - Process svn:externals definitions ( ).

The options for ignored and deleted files are switched on and off from *the Settings menu* of the **Working Copy** panel:

- the format of the currently loaded working copy.
- the current numbers of incoming changes ( ), outgoing changes ( ) and conflicting changes ( ).
- a progress bar for the currently running SVN operation and a button ( ) that allows you to stop it.

## Getting Started

This section explains the basic operations that can be done in Syncro SVN Client.

### SVN Repository Location

This section explains how to add and edit the repository locations in Syncro SVN Client.

#### Add / Edit / Remove Repository Locations

Usually, team members do all of their work separately, in their own working copy, and then must share their work by committing their changes. This is done using an Apache Subversion™ repository. Oxygen XML Editor supports versions 1.4, 1.5, 1.6, 1.7, and 1.8 of the SVN repository format.

Before you can begin working with a Subversion repository, you must define a repository location in the [Repositories view](#).

To create a repository location, use the  **New Repository Location...** action that is available in the **Repository** menu, the **Repositories** view toolbar, and in the contextual menu. This action opens the **New Repository Location** dialog box, which prompts you for the [URL of the repository](#) you want to connect to. You can also [use peg revisions at the end of the URLs](#) (for example, URL@rev1234) to browse only that specific revision. No authentication information is requested at the time the location is defined. It is left to the Subversion client to request the user and password information when it is needed. The main benefit of allowing Subversion to manage your password is that it prompts you for a new password only when your password changes.

Once you enter the repository URL, Oxygen XML Editor tries to contact the server to get the content of the repository for displaying it in the [Repositories view](#). If the server does not respond in the timeout interval set in the preferences, an error is displayed. If you do not want to wait until the timeout expires, you can use the  **Stop** button from the toolbar of the view.

To edit a repository location, use the  **Edit Repository Location...** action that is available in the **Repository** menu and in the contextual menu. This action opens the **Edit Repository Location** dialog box, which prompts you for the [URL of the repository](#) you want to connect to. You can also [use peg revisions at the end of the URLs](#) (for example, URL@rev1234) to browse only that specific revision.

To remove a repository location, use the  **Remove Repository Location...** action that is available in the **Repository** menu and in the contextual menu. A confirmation dialog box is displayed to make sure that you do not accidentally remove the wrong locations.

The order of the repositories can be changed in the **Repositories** view at any time with the  **Up** arrow and  **Down** arrow buttons on the toolbar of the view. For example, pressing the up arrow once moves the selected repository in the list up one position.

To set the reference revision number of an SVN repository use the **Change the Revision to Browse...** action that is available in the **Repository** menu and in the contextual menu. The revision number of the repository is used for displaying the contents of the repository when it is viewed in the [Repositories view](#). Only the files and folders that were present in the repository at the moment when this revision number was generated in the repository are displayed as contents of the repository tree. Also, this revision number is used for all the operations executed directly from the [Repositories view](#).

### Authentication

Five protocols are supported: *HTTP*, *HTTPS*, *SVN*, *SVN + SSH* and *FILE*. If the repository that you are trying to access is password protected, the **Enter authentication data** dialog box requests a user name and a password. If the **Store authentication data** checkbox is checked, the credentials are stored in Apache Subversion™ default directory:

- on Windows - %HOME%\Application Data\Subversion\auth. Example: C:\Documents and Settings\John\Application Data\Subversion\auth
- on Linux and OS X - \$HOME/.subversion/auth. Example: /home/John/.subversion/auth

There is one file for each server that you access. If you want to make Subversion forget your credentials, you can use the **Reset authentication** command from the **Options** menu. This causes Subversion to forget all your credentials. When you reset the authentication data, restart Oxygen XML Editor for the change to take effect.

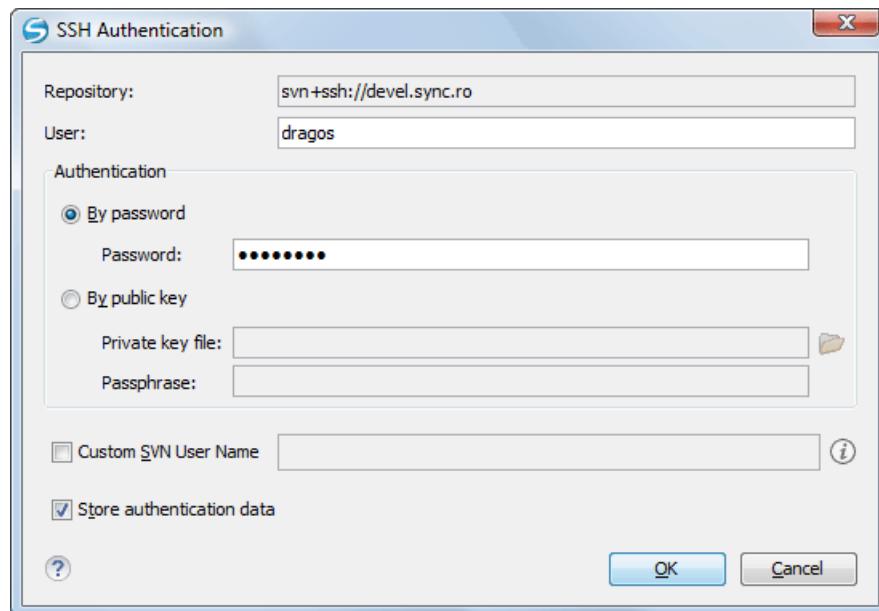


**Tip:** The *FILE* protocol is recommended if the SVN repository and Oxygen XML Editor are located on the same computer as it ensures faster access to the SVN repository compared with other protocols.

For HTTPS connections where client authentication is required by your SSL server, you must choose the certificate file and enter the corresponding certificate password which is used to protect your certificate.

When using a secure HTTP (HTTPS) protocol for accessing a repository, a **Certificate Information** dialog box is displayed and asks you whether you want to accept the certificate permanently, temporarily, or simply deny it.

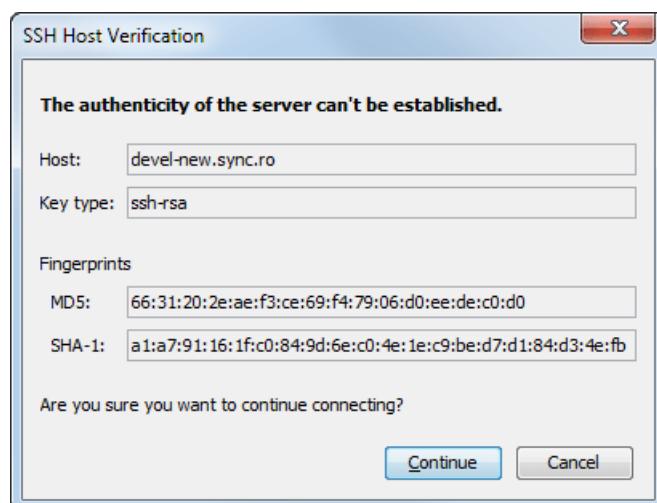
If the repository has SVN+SSH protocol, the SSH authentication can also be made with a private key and a pass phrase.



**Figure 353: User & Private Key Authentication Dialog Box**

After the SSH authentication dialog box, another dialog box appears for entering the SVN user name that accesses the SVN repository. The SVN user name is recorded as the *committer* in SVN operations.

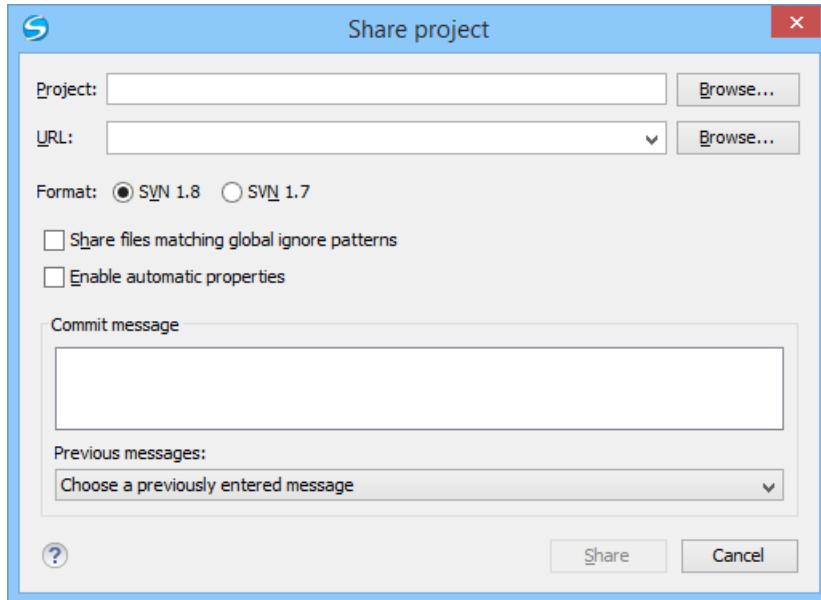
When connecting for the first time to a Subversion repository through SVN+SSH protocol, you will be asked to confirm if you trust the SSH host. The same dialog box is also displayed when the server changed the SSH key or when the key was deleted from the local Subversion cache folder.



**Figure 354: SSH server name and key fingerprint**

## Share a Project

Even if you start developing a new project, or you want to migrate an existing one to Subversion, the Syncro SVN Client allows you to easily share it with the rest of your team. The shared project directory is automatically converted to a working copy and added under Syncro SVN Client management. The **Share project...** action is available in the **Tools** menu and the contextual menu of the **Repositories** view.



**Figure 355: Share Project Dialog Box**

The following options can be configured in the **Share project** dialog box:

### Project

*The location of the project folder* on the local disk by using the text box or the **Browse** button. This folder should not be empty or already under version control.

**Important:** By default, the SVN system only imports the content of the specified folder, and not the root folder itself. Therefore, it is recommended to use the **Browse** button to select the project folder so that the client will automatically append the name of it to the specified URL.

### URL

*The new location of the project* (inside the repository) that will be used to access it.

**Note:** *Peg revisions* have no effect for this operation since it is used to send information to the repository.

**Attention:** If the new location already exists, make sure that it is an empty directory to avoid mixing your project content with other files (if items exist with the same name, an error will occur and the operation will not proceed). Otherwise, if the address does not exist, it is created automatically.

### Format

The SVN format of the working copy. You can choose between **SVN 1.8** or **SVN 1.7**.

### Share files matching global ignore patterns

When enabled, the file names that match the patterns defined in either of the following locations are also imported into the repository:

- The *global-ignores* property in *the SVN configuration file*.
- The *File name ignore patterns* option in the *SVN > Working Copy preferences page*.

## Enable automatic properties/Disable automatic properties

Enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in [the SVN configuration file](#).

 **Note:** This option is available only when there are defined properties to be applied automatically for newly added items under version control. You can define these properties in the SVN config file (in the `auto-props` section). Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable automatic properties**, or **Disable automatic properties**.

## Defining a Working Copy

An Apache Subversion™ working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, your working copy being your private work area. In order to make your own changes available to others or incorporate other people's changes, you must explicitly tell Subversion to do so. You can even have multiple working copies of the same project.

Name	Date	Revision	Author	Size	Type
E:\svnkit					File Folder
gradle					File Folder
wrapper					File Folder
gradle-wrapper.jar	May 4, 2011	7623	alex		Executable ...
gradle-wrapper.properties	May 4, 2011	7623	alex		PROPERTIE...
svnkit	May 4, 2011	7618	alex		File Folder
svnkit-dl	May 4, 2011	7623	alex		File Folder
.settings	May 15, 2011	7636	alex		File Folder
src	May 10, 2011	7630	alex		File Folder
main	May 4, 2011	7618	alex		File Folder
conf	May 10, 2011	7630	alex		File Folder
java	May 4, 2011	7618	alex		File Folder
resources	May 4, 2011	7622	alex		File Folder
scripts	May 4, 2011	7618	alex		File Folder
jsvn	May 10, 2011	7630	alex		File Folder
jsvn.bat	May 4, 2011	7618	alex	2 KB	File
jsvnsetup.openvms	May 4, 2011	7618	alex	2 KB	Windows B...
build.gradle	May 4, 2011	7618	alex	1 KB	OPENVMS File
svnkit-dav	May 4, 2011	7620	alex	2 KB	GRADLE File
svnkit-distribution	May 4, 2011	7623	alex		File Folder
svnkit-javahl16	May 4, 2011	7618	alex		File Folder
svnkit-osgi	May 4, 2011	7623	alex		File Folder
svnkit-test	May 4, 2011	7635	alex		File Folder
.settings	May 12, 2011	7635	alex		File Folder
configurations	May 4, 2011	7618	alex		File Folder

**Figure 356: Working Copy View**

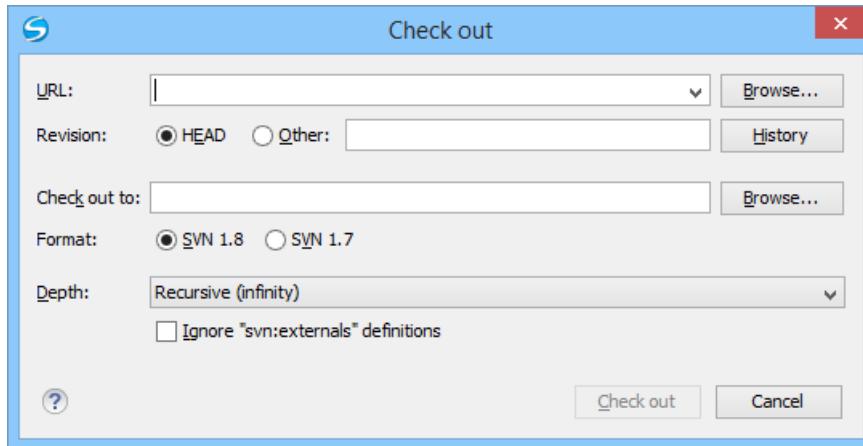
A Subversion working copy also contains some extra files, created and maintained by Subversion, to help it keep track of your files. In particular, each directory in your working copy contains a subdirectory named `.svn`, also known as the working copy *administrative directory*. This administrative directory contains an unaltered copy of the last updated files from the repository. This copy is usually referred to as the *pristine copy* or the *BASE revision* of the working copy. These files help Subversion recognize which files contain unpublished changes, and which files are out-of-date with respect to others' work.

A typical Subversion repository often holds the files (or source code) for several projects. Usually each project is a subdirectory in the repository's file system tree. In this arrangement, a user's working copy usually corresponds to a particular sub-tree of the repository.

## Check Out a Working Copy

*Check out* means to make a copy of a project from a repository to your local file system. This copy is called a *working copy*. An Apache Subversion™ working copy is a specially formatted directory structure that contains additional .svn directories, which store Subversion information, as well as a pristine copy of each item that is checked out.

To check out a working copy, locate and select the desired directory in the **Repositories** view and select the **Check out...** action from the contextual menu, the toolbar, or the **Repository** menu.



**Figure 357: Check Out Dialog Box**

The following options can be configured in the **Check out** dialog box:

### URL

*The location of the repository directory to be checked out.*

- Note:** To check out an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, URL@rev1234).

### Revision

You can choose between the **HEAD** or **Other** revision. If you need to *check out* a specific revision, specify it in the **Other** text box or use the **History** button and choose a revision from *the History dialog box*.

### Check out to

Specify *the location where you want to check out* the new working copy by typing the local path in the text box or by using the **Browse** button. If the specified local path does not point to an existing directory, it will automatically be created.

- Important:** By default, the SVN system only checks out the content of the directory specified by the URL, and not the directory itself. Therefore, it is recommended to use the **Browse** button to select the *check out* location so that the client will automatically append the name of the remote directory to the path of the selected directory.

- Warning:** The destination directory should be empty. If files exist, they are skipped (left unchanged) by the *check out* operation and *displayed as modified* after the operation has finished. Also, the destination directory must not already be under version control.

### Format

The SVN format of the working copy. You can choose between **SVN 1.8** or **SVN 1.7**.

### Depth

The depth is useful if you want to *check out* only a part of the selected repository directory and bring the rest of the files and subdirectories in a future update. You can find out more about the checkout depth in the *sparse checkouts* section. You can choose between the following depths:

- **Recursive (infinity)** - Checks out all the files and folders contained in the selected folder.
- **Immediate children (immediates)** - Checks out only the child files and folders without recursing subfolders.
- **File children only (files)** - Checks out only the child files.
- **This folder only (empty)** - Checks out only the selected folder (no child file or folder is included).

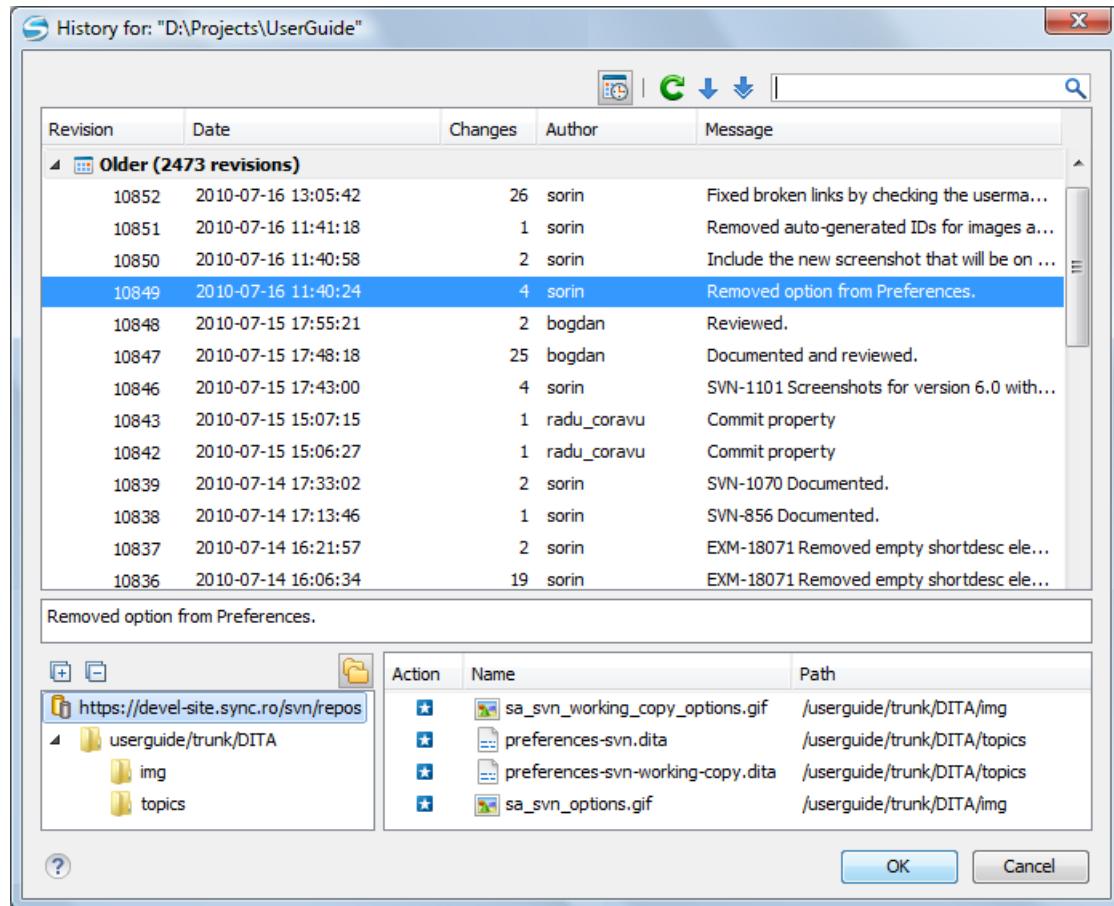
#### Ignore "svn:externals" definitions

When enabled, external items are ignored in the *check out* operation. This option is only available if you choose the **Recursive (infinity)** depth.

After a check out, the new working copy is added to the list in the [Working Copy view](#) and loaded automatically.

#### The History Dialog Box

The **History** dialog box presents a list of revisions for a resource. It is opened from the dialog boxes that require setting an SVN revision number, such as [the Check Out dialog box](#) or [the Branch / Tag dialog box](#). It presents information about revision, commit date, author, and commit comment.



**Figure 358: History Dialog Box**

The initial number of entries in the list is 50. Additional revisions can be added to the list using the **Get next 50** and **Get all** buttons. The list of revisions can be refreshed at any time with the **Refresh** button. You can group revisions in predefined time frames (today, yesterday, this week, this month), by pressing the **Group by date** button from the toolbar.

The **Affected Paths** area displays all paths affected by the commit of the revision selected in history. You can see the changes between the selected revision and the file's previous state using the **Compare with previous version** action, available in the contextual menu.

## Use an Existing Working Copy

Using an existing working copy is the process of taking a working copy that exists on your file system and connecting it to Apache Subversion™ repository. If you have a brand new project that you want to import into your repository, then see the section [Import resources into the repository](#). The following procedure assumes that you have an existing valid working copy on your file system.

1. Click the **Working Copies Manager** toolbar button  (on Mac OS X) in the [Working Copy view](#).

This action opens the **Working copies list** dialog.

2. Press the **Add** button.
3. Select the working folder copy from the file system. The name is useful to differentiate between working copies located in folders with the same name. The default name is the name of the root folder of the working copy.

 **Note:** For SVN 1.7 and newer working copies, all the internal information is kept only in the root directory. Thus, Syncro SVN Client needs to load the whole working copy.

4. Press the **OK** button.

The selected working copy is loaded and presented in the [Working Copy view](#).

 **Notice:** You can add working copies older than SVN 1.7. However, to load any of them, Syncro SVN Client will require to upgrade the working copy to SVN 1.8 format.

## Manage Working Copy Resources

This section explains how to work with the resources that are displayed in the [Working Copy view](#).

### Edit Files

You can edit files from the [Working Copy view](#) by double clicking them or by right clicking them and choosing **Open** from the contextual menu.

Please note that only one file can be edited at a time. If you try to open another file, it is opened in the same editor window. The editor has syntax highlighting for known file types, meaning that a different color is used for each type of recognized token in the file. If the selected file is an image, then it is previewed in the editor, with no access to modifying it.

After modifying and saving a file from a working copy, a modified marker - an asterisk (\*) - will be added to the file's icon in the [Working Copy view](#). The asterisk marks the files that have local modifications that were not committed to the repository.

### Add Resources to Version Control

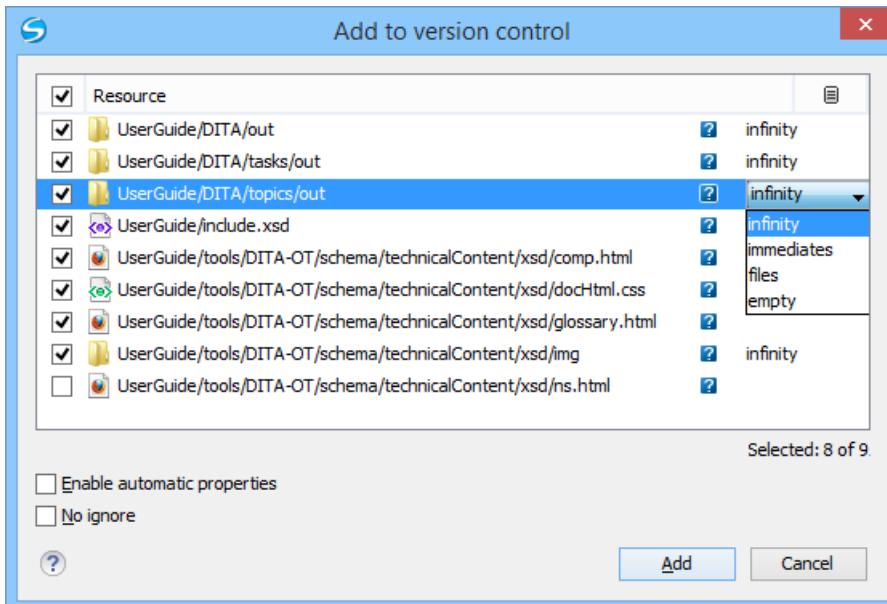
To share new files and folders (created in your working copy), add them to version control using the **Add to version control** option from the [Working Copy view](#).

You can easily spot resources not under version control by the  (*unversioned*) icon displayed in the  **Local file status** column. Resources scheduled for addition (*added*) are displayed with this icon  in the [Working Copy view](#) and are added in the repository after you commit them.

 **Note:** Do not make a confusion between  and  icons. The former icon stands for resources that are actually copies of resources already committed in the repository, meaning they are *scheduled for addition with history*.

When you use the **Add to version control** option on a directory, its entire structure is scanned and all the resources that can be added under version control are presented.

Though it is not mandatory to add resources under version control explicitly, it is recommended. If you forgot to add a resource, when you [commit your changes](#), the resource is presented in the commit dialog box, but not selected. When you commit an *unversioned* resource, it is automatically added under version control before starting the commit operation.



**Figure 359: Add to Version Control Dialog Box**



**Note:** Ignored (■) items can also be added under version control.

The **Depth** column is displayed only when directories are also presented in the dialog box. For any directory, you can use one of the available values to instruct Subversion to limit the scope of the operation to a particular tree depth.



**Note:** The initial value of the **Depth** field can have the following values, depending on the [listing mode of the items in the working copy view](#):

- *infinity* - When the working copy items are presented as a tree.
- *files* - When the working copy items are presented compressed.
- *empty* - When the working copy items are presented flat.

When you add unversioned or ignored directories, the initial value of the **Depth** field also depends on the state of the **Show unversioned directories content** and **Show ignored directories content** options. In case these options are enabled, the value is based on the listing mode of the items in the working copy view. When they are disabled, the value is *empty*.

The following options are available in this dialog box:

- **Enable automatic properties** or **Disable automatic properties** - enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in the `config` file of the Subversion configuration directory.



**Note:** This option is available only when there are defined properties to be applied automatically for resources newly added under version control. You can define these properties in the `config` file of the Subversion configuration directory, in the `auto-props` section. Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable automatic properties**, or **Disable automatic properties**.

- **No ignore** - when you enable this option, file-name patterns defined to ignore *unversioned* resources do not apply. Resources that are located inside an *unversioned* directory selected for addition, and match these patterns, are also scheduled for addition in the repository.



**Note:** This option is available only when directories are also presented in the dialog box.

You can define file-name patterns to ignore *unversioned* resources in one of the following locations:

- In the `config` file of the Subversion configuration directory (the `global-ignores` option from the `miscellany` section).
- In the Oxygen XML Editor options ([open the Preferences dialog box](#) and go to **SVN > Working copy > Application global ignores**).

Each of the above two options is activated only when you select an item for which the option can be applied.

### Ignore Resources Not Under Version Control

Some resources inside your working copy do not need to be subject to version control. These resources can be files created by the compiler, `*.obj`, `*.class`, `*.lst`, or output folders used to store temporary files. Whenever you [commit changes](#), Apache Subversion™ shows your modified files but also the unversioned files, which fill up the file list in the commit dialog box. Though the unversioned files are committed unless otherwise specified, it is difficult to see exactly what you are committing.

The best way to avoid these problems is to add the derived files to the Subversion's ignore list. That way they are never displayed in the commit dialog box and only genuine unversioned files which must be committed are shown.

You can choose to ignore a resource by using the **Add to svn:ignore** action in the contextual menu of the [Working Copy view](#).

In the **Add to svn:ignore** dialog box, you can specify the resource to be ignored by name or by a custom pattern. The custom pattern can contain the following wildcard characters:

- \* - Matches any string of characters of any size, including the empty string.
- ? - Matches any single character.

For example, you can choose to ignore all text documents by using the pattern: `*.txt`.

The action **Add to svn:ignore** adds a predefined Subversion property called `svn:ignore` to the parent directory of the specified resource. In this property, there are specified all the child resources of that directory that must be ignored. The result is visible in the **Working Copy** view. The ignored resources are represented with grayed icons.

### Delete Resources

The **Delete** action is available in the contextual menu of the [Working Copy view](#). When you delete an item from the working copy, it is marked as *deleted* (scheduled for deletion from repository upon the next commit) and removed from the file system. Depending on the state of each item, you are prompted to confirm the operation.

If a resource is deleted from the file system without Subversion's knowledge, the resource is marked as *missing* () in your working copy. You can decide what you want to do with a *missing* item:

- in case of a commit, any *missing* item is first deleted automatically and then committed.
- Note:** Not any *missing* item can be committed as *deleted*, and removed from the repository. For example, you cannot commit an item that no longer exists on the disk and that was scheduled for addition () previously, since this item does not exist in the repository, but you can use the **Delete** action instead.
- in case you want to recover *missing* items, either [update](#) the items themselves or one of their parent directories. This fetches their latest version from the repository.

You can also delete conflicting items (file content conflicts, property conflicts, tree-conflicts) and Syncro SVN Client automatically marks them as resolved.

**Note:** It is recommended that you resolve conflicts manually to avoid losing any important remote modifications.

Finally, you can change your mind and [revert](#) the deleted items to their initial, pristine, state.

## Copy Resources

You can copy several resources from different locations of the working copy. You select them in the [Working Copy view](#) and then use **Copy to** from the contextual menu. This is not a simple file system copy, but an Apache Subversion™ command. It will copy the resource and the copy will also have the original resource's history. This is one of Subversion's very important features, as you can keep track of where the copied resources originated.

Based on the selected items, the **Copy to** action is enabled only if it can be performed. Even if the operation would not normally be possible in SVN (due to some invalid local file states against copy), Oxygen XML Editor performs the copy operation as a simple file system operation. This means no SVN versioning meta-data is affected.



### Note:

- In case you copy an item to a directory that is *not under version control* (*unversioned* or *ignored*), the history of the item is not preserved. For example, when copying directories, all items inside them will also be copied without history.
- In case you copy a directory that contains *external* items, these are not copied. This is specific for SVN 1.7 working copies only. To fetch the *external* items, use the **Update** operation on the copied directory.

In the **Copy to** dialog box, you can navigate through the working copy directories in order to choose a target directory, to copy inside it. If you try to copy a single resource you are also able to change that resource's name. For *versioned* items, you can select **Ignore resource history** to copy them without their history (similar to a simple file system copy).



**Note:** The **Copy to** dialog box only presents all the local directories that are a valid destination against the copy operation, based on their local file status. Also, the [working copy settings](#) are taken into account.

In the **Commit** dialog box, only the directory in question will appear without its children.

## Move Resources

As in the case of the copy command, you can move several resources at once. Select the resources in the [Working Copy view](#) and choose the **Move to** action from the contextual menu. The move command actually behaves as if a copy followed by a delete command were issued. You will find the moved resources at the desired destination and also at their original location, but marked as *deleted*.



**Note:** *External* items cannot be moved using the **Move to** action, because they cannot be deleted. Instead, you should edit the `svn:externals` property defining the *external* item and use the **Update** operation on the item's parent folder for the change to take effect.



**Attention:** For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

## Rename Resources

The **Rename** action is available in the contextual menu of the [Working Copy view](#) and can be performed on a single resource. This action acts as a move command with the destination directory being the same as the original location of the resource. A copy of the original item is created with the new name, also keeping its history. The original item is marked as *deleted*.



**Note:** *External* items cannot be renamed using the **Rename** action because they cannot be deleted. Instead, you should edit the `svn:externals` property defining the *external* item, then use the **Update** operation on the item's parent folder for the change to take effect.



**Attention:** For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

## Lock / Unlock Resources

The idea of version control is based on the *copy-modify-merge* model of file sharing. This model states that each user contacts the repository and creates a local working copy (check out). Users can then work independently and modify their working copies as they please. When their goal has been accomplished, it is time for the users to share their work

with the others, to send them to the repository (commit). When a user has modified a file that has been also modified on the repository, the two files will have to be merged. The version control system assists the user with the merging as much as it can, but in the end the user is the one that must make sure it is done correctly.

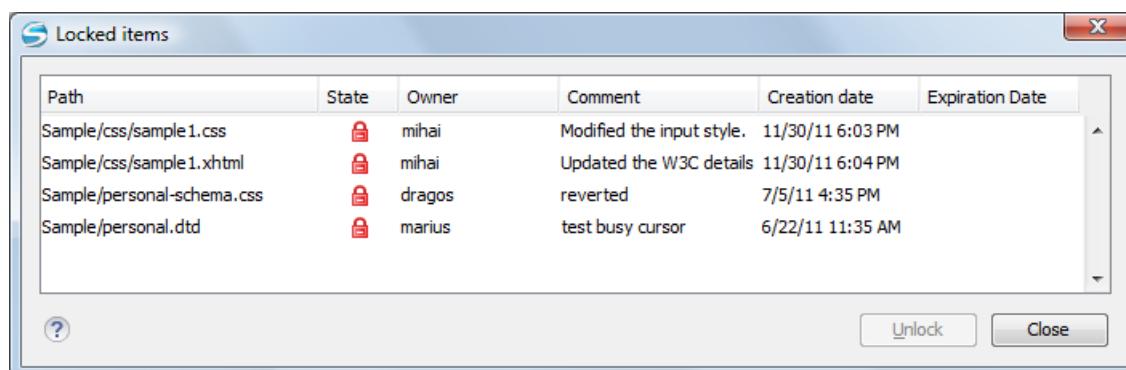
The copy-modify-merge model only works when files are contextually mergeable: this is usually the case of line-based text files (such as source code). However this is not always possible with binary formats, such as images or sounds. In these situations, the users must each have exclusive access to the file, ending up with a *lock-modify-unlock* model. Without this, one or more users could end up wasting time on changes that cannot be merged.

An SVN lock is a piece of metadata which grants exclusive access to a user. This user is called the lock owner. A lock is uniquely identified by a lock token (a string of characters). If someone else attempts to commit the file (or delete a parent of the file), the repository demands two pieces of information:

- User authentication - the user performing the commit must be the lock owner
- Software authorization - the user's working copy must have the same lock token as the one from the repository, proving that it is the same working copy where the lock originated from.

### *Scanning for Locks*

When starting to work on a file that is not contextually mergeable (usually a binary file), it is better to verify if someone else is not already working on that file. You can do this in the [Working Copy view](#) by selecting one or more resources, then right clicking on them and choosing the **Scan for Locks** action from the context menu.



**Figure 360: The locked items dialog**

The **Locked items** dialog contains a table with all the resources that were found locked on the repository. For each resource there are specified: resource path, state of the lock, owner of the lock, lock comment, creation and expiration date for the lock (if any).

The state of the lock can be one of:

- 🔒 - shown when:
  - another user has locked the file in the repository;
  - the file was locked by the same user from another working copy;
  - the file was locked from the **Repositories** view.
- 🔓 - displayed after you have locked a file from the current working copy.
- 🔑 - a file already locked from your working copy is no longer locked in the repository (it was unlocked by another user).
- 🔐 - a file already locked from your working copy is being locked by another user. Now the owner of the file lock is the user who stole the lock from you.

You can unlock a resource by selecting it and pressing the **Unlock** button.

### *Locking a File*

By locking a file, you have exclusive write access to it in the repository.

You can lock a file from your working copy or directly from the **Repositories** view.



**Note:** You can only lock files (not directories). This is a restriction imposed by Apache Subversion™.

The **Lock** dialog box allows you to write a comment when you set a lock or when you *steal* an existing one. Note that you should *steal* a lock only after you made sure that the previous owner no longer needs it, otherwise you may cause an unsolvable conflict, which could be the reason the lock was put there in the first place. The Subversion server can have a policy concerning lock stealing, as it may not allow you to do this if certain conditions are not met.

The lock stays in place until you unlock the file or until someone breaks it. There is also the possibility that the lock expires after a period of time specified in the Subversion server policy.

### *Unlocking a File*

A file can be unlocked from the contextual menu of the [Working Copy view](#). A dialog will prompt you to confirm the unlocking and it will also allow you to break the lock (unlock it by force).

### **Synchronize with Repository**

In the work cycle you will need to incorporate other people's changes (update) and to make your own work available to others (commit). This is what the **Incoming** and **Outgoing** modes of [the Working Copy view](#) was designed for, to help you send and receive modifications from the repository.

The **Incoming** and **Outgoing** modes of this view focus on incoming and outgoing changes. The incoming changes are the changes that other users have committed in the repository since you last updated your working copy. The outgoing changes are the modifications you made to your working copy as a result of editing, removing or adding resources.

The view presents the status of the working copy resources against the BASE revision after a **Refresh** operation. You can view the state of the resources versus a repository HEAD revision by using the **Synchronize** action from [the Working Copy view](#).

### **View Differences**

One of the most common requirements in project development is to see what changes have been made to the files from your Working Copy or to the files from the repository. You can examine these changes after a synchronize operation with the repository, by using the **Open in compare editor** action from the contextual menu.

The text files are compared using a built-in [Compare view](#) which uses a line differencing algorithm or a specified external diff application if such an application is [set in the SVN preferences](#). When a file with outgoing status is involved, the compare is performed between the file from the working copy and the BASE revision of the file. When a file with incoming or conflict status is involved, the differences are computed using a three-way algorithm which means that the local file and the repository file are each compared with the BASE revision of the file. The results are displayed in the same view. The differences obtained from the local file comparison are considered outgoing changes and the ones obtained from the repository file comparison are considered incoming changes. If any of the incoming changes overlap outgoing changes then they are in conflict.

A special case of difference is a *diff pseudo-conflict*. This is the case when the left and the right sections are identical but the BASE revision does not contain the changes in that section. By default this type of changes are ignored. If you want to change this you can go to [SVN Preferences](#) and change the corresponding option.

The right editor of the internal compare view presents either the BASE revision or a revision from the repository of the file so its content cannot be modified. By default when opening a synchronized file in the **Compare** view, a compare is automatically performed. After modifying and saving the content of the local file presented in the left editor, another compare is performed. You will also see the new refreshed status in the [Working Copy view](#).

```

E:\NewSamples\personal.css*
personal.css
11 border-bottom: 0.1em solid navy;
12 padding: 0.3em;
13
14 font-size:large;
15 font-weight:bold;
16
17 color:navy;
18 background-color:inherit;
19
20 }
21
22 personnel{
23 display:block;
24 margin:1em;
25 /*Counter for the person elements*/
26 counter-reset:pers_cnt;
27 }
28
29 person{
30 display:block;
31
32 margin: 1em;
33 font-size:medium;
34 font-weight:normal;
35 border:0.1em solid #DDDDEE;
36 padding:0.5em;
37
38 color:inherit;
39 background-color: #EFEFEE;
40 }

personal.css@HEAD [dragos]
}
personnel:before{
 display:block;
 content:"List of employees";
}

border-bottom: 0.4em solid navy;
padding: 0.2em;
font-size:small;
font-weight:bold;
color:blue;
background-color:inherit;
}

person{
 display:block;
 margin: 2em;
 border:0.3em solid #DDDDEE;
 padding:0.2em;
 color:inherit;
 background-color: #EFECCC;
 /*Increments the person counter.*/
 counter-increment:pers_cnt;
}

name,
family,
given,

```

**Figure 361: Compare View**

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

There are three types of differences:

- incoming changes - Changes committed by other users and not present yet in your working copy file. They are marked with a blue highlight and on the middle divider the arrows point from right to left.
- outgoing changes - Changes you have done in the content of the working copy file. They are marked with a gray highlight and the arrows on the divider are pointing from left to right.
- conflicting changes - This is the case when the same section of text which you already modified in the local file has been modified and committed by some other person. They are marked with a red highlight and red diamonds on the divider.

There are numerous actions and options available in the [Compare View toolbar](#) or in the **Compare** menu from the main menu. You can decide that some changes need adjusting or that new ones must be made. After you perform the adjustments, you may want to perform a new compare between the files. For this case there is an action called **Perform files differencing**. After each files differencing operation the first found change will be selected. You can navigate from one change to another by using the actions **Go to first**, **Go to previous**, **Go to next** and **Go to last modification**. If you decide that some incoming change needs to be present in your working file you can use the action **Copy change from right to left**. This is useful also when you want to override the outgoing modifications contained in a conflicting section. The action **Copy all non-conflicting changes from right to left** copies all incoming changes which are not contained inside a conflicting section in your local file.

Let us assume that only a few words or letters are changed. Considering that the differences are performed taking into account whole lines of text, the change will contain all the lines involved. For finding exactly what words or letters have

changed there are available two dialogs which present a more detailed compare result when you double click on the middle divider of a difference: **Word Details** and **Character Details**.

When you want to examine only the changes in the real text content of the files disregarding the changes in the number of white spaces between words or lines there is available an option in the [SVN Preferences](#) which allows you to enable or disable the white space ignoring feature of the compare algorithm.

## Conflicts

A file conflict occurs when two or more developers have changed the same few lines of a file or the properties of the same file. As Subversion knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and try to analyse and resolve the conflicting situation.

### Real Conflicts vs Mergeable Conflicts

There are two types of conflicts:

- *real conflict* ( decorator in *Name* column) - Syncro SVN Client considers the following resource states to be real conflicts:
  - *conflicted* state - a file reported by SVN as being in this state is obtained after it was updated/merged while having incoming and outgoing content or property changes at the same time, changes which could not be merged. A content conflict ( symbol in *Local file status* column) is reported when the modified file has binary content or it is a text file and both local and remote changes were found on the same line. A properties conflict ( symbol in *Local properties status* column) is reported when a property's value was modified both locally and remotely;
  - *tree conflicted* state ( symbol in *Local file status* column) - obtained after an update or merge operation, while having changes at the directory structure level (for example, file is locally modified and remotely deleted or locally scheduled for deletion and remotely modified);
  - *obstructed* state ( symbol in *Local file status* column) - obtained after a resource was versioned as one kind of object (file, directory, symbolic link), but has been replaced outside Syncro SVN Client by a different kind of object.
- *pseudo-conflict* ( decorator in *Name* column) - a file is considered to be in *pseudo-conflict* when it contains both incoming and outgoing changes. When incoming and outgoing changes do not intersect, an update operation may automatically merge the incoming file content into the existing locally one. In this case, the *pseudo-conflict* marker is removed. This marker is used only as a warning which should prevent you to run into a real conflict.



#### Note:

- A conflicting resource cannot be committed to repository. You have to resolve it first, by using **Mark Resolved** action (after manually editing/merging file contents) or by using **Mark as Merged** action (for pseudo-conflicts).
-  and  decorators are presented only when one of the following view modes is selected: **Modified, Incoming, Outgoing, Conflicts**.
- The  marker is used also for folders to signal that they contain a file in real conflict or pseudo-conflict state.

### Content Conflicts vs Property Conflicts

A *Content conflict* appears in the content of a file. A merge occurs for every inbound change to a file which is also modified in the working copy. In some cases, if the local change and the incoming change intersect each other, Apache Subversion™ cannot merge these changes without intervention. So if the conflict is real when updating the file in question the conflicting area is marked like this:

```
<<<<< filename
your changes
=====
code merged from repository
```

&gt;&gt;&gt;&gt;&gt; revision

Also, for every conflicted file Subversion places three additional temporary files in your directory:

- `filename.ext.mine` - This is your file as it existed in your working copy before you updated your working copy, that is without conflict markers. This file has your latest changes in it and nothing else.
- `filename.ext.rOLDREV` - This is the file that was the BASE revision before you updated your working copy, that is the file revision that you updated before you made your latest edits.
- `filename.ext.rNEWREV` - This is the file that Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD revision of the repository.

OLDREV and NEWREV are revision numbers. If you have conflicts with binary files, Subversion does not attempt to merge the files by itself. The local file remains unchanged (exactly as you last changed it) and you will get `filename.ext.r*` files also.

A *Property conflict* is obtained when two people modify the same property of the same file or folder. When updating such a resource a file named `filename.ext.prej` is created in your working copy containing the nature of the conflict. Your local file property that is in conflict will not be changed. After resolving the conflict you should use the **Mark resolved** action in order to be able to commit the file. Note that the **Mark resolved** action does not really resolve the conflict. It just removes the conflicted flag of the file and deletes the temporary files.

#### Edit Real Content Conflicts

The conflicts of a file in the conflicted state (a file with the red double arrow icon) can be edited visually with the **Compare** view (the built-in file diff tool) or with an *external diff application*. Resolving the conflict means deciding for each conflict if the local version of the change will remain or the remote one instead of the special conflict markers inserted in the file by the SVN server.

The **Compare** view (or the external diff application *set in Preferences*) is opened with the action **Edit Conflict** which is available on the contextual menus of *the Working Copy view* and is enabled only for files in the conflicted state (an update operation was executed but the differences could not be merged without conflicts). The external diff application is called with 3 parameters because it is a 3-way diff operation between the local version of the file from the working copy and the HEAD version from the SVN repository with the BASE version from the working copy as common ancestor.

If *the option Show warning dialog when edit conflicts is enabled* you will be warned at the beginning of the operation that the operation will overwrite the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<, =====, >>>>>) with the original local version of the file that preceded the update operation. If you press the OK button the visual conflict editing will proceed and a backup file of the conflict version received from the SVN server is created in the same working copy folder as the file with the edited conflicts. The name of the backup file is obtained by appending the extension `.sync.bak` to the file as stored on the SVN server. If you press the **Cancel** button the visual editing will be aborted.

The usual actions on the differences between two versions of a file are available on the toolbar of this view:



#### Save

Saves the modifications of the local version of the file displayed in the left side of the view.



#### Perform Files Differencing

Performs a comparison between the source file and target file.



#### Ignore Whitespaces

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.



#### Synchronized scrolling

Synchronizes scrolling so that a selected difference can be seen on both sides of the application window. This action enables/disables the previously described behavior.

### **Format and Indent Both Files (Ctrl Shift P (Command Shift P on OS X))**

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.



#### **Copy Change from Right to Left**

Copies the selected difference from the target file in the right side to the source file in the left side.



#### **Copy All Changes from Right to Left**

Copies all changes from the target file in the right side to the source file in the left side.



#### **Next Block of Changes (Ctrl . (Command . on OS X))**

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes.



**Note:** A change block groups one or more consecutive lines that contain at least one change.



#### **Previous Block of Changes (Ctrl , (Command , on OS X))**

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes.



#### **Next Change (Ctrl Shift . (Command Shift . on OS X))**

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change or when there are no changes.



#### **Previous Change (Ctrl Shift , (Command Shift , on OS X))**

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change or when there are no changes.



#### **First Change (Ctrl B (Command B on OS X))**

Jumps to the first change.

The operation begins by overwriting the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<, =====, >>>>>) with the original local version of the file before running the update action which created the conflict. After that the differences between this original local version and the repository version are displayed in the **Compare** view.

If you want to edit the conflict version of the file directly in a text editor instead of the visual editing offered by the **Compare** view you should work on the local working copy file after the update operation without running the action **Edit Conflict**. If you decide that you want to edit the conflict version directly after running the action **Edit Conflict** you have to work on the `.sync.bak` file.

If you did not finish editing the conflicts in a file at the first run of the action **Edit Conflict** you can run the action again and you will be prompted to choose between resuming the editing where the previous run left it and starting again from the conflict file received from the SVN server.

After the conflicts are edited and saved in the local version of the file you should run:

- either the action **Mark Resolved** on the file so that the result of the conflict editing process can be committed to the SVN repository,
- or the action **Revert** so that the repository version overwrites all the local modifications.

Both actions remove the backup file and other temporary files created with the conflict version of the local file.

#### **Revert Your Changes**

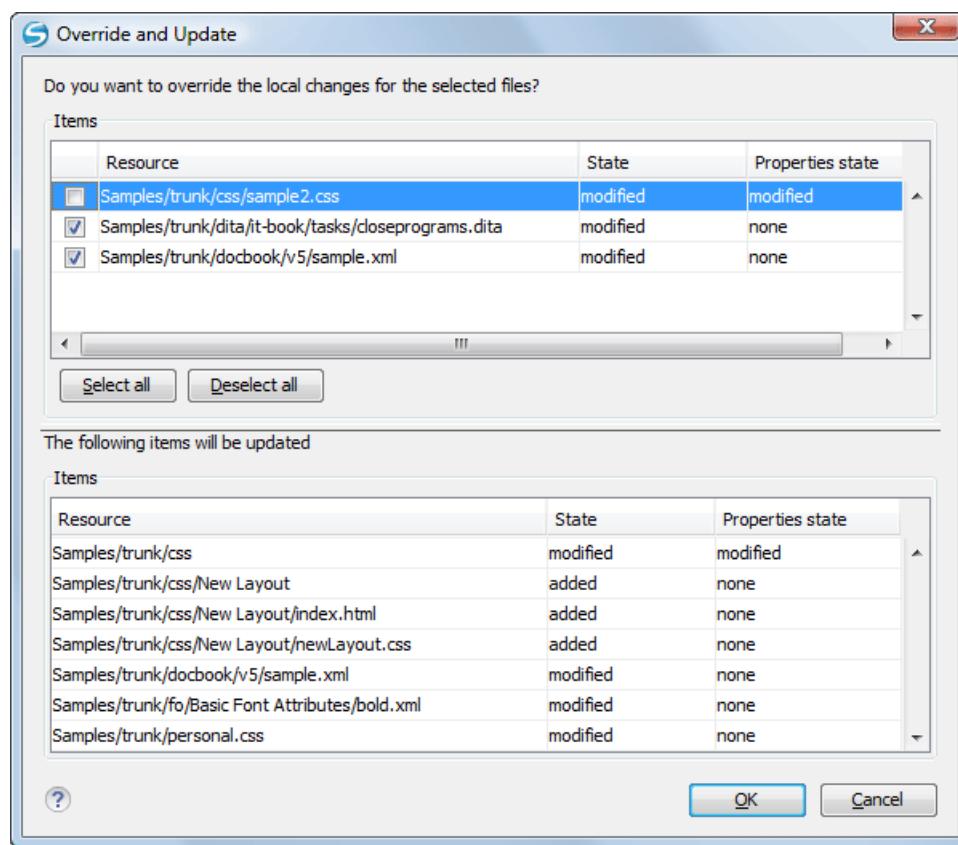
If you want to undo changes made in your working copy, since the last update, select the items you are interested in, right click to display the contextual menu and select **Revert**. A dialog will pop up showing you the files and folders that

you have changed and can be reverted. Select those you want to revert and click the **OK** button. Revert will undo only your local changes. It does not undo any changes which have already been committed. If you choose to revert a conflicting item to its pristine copy, then the eventual conflict is solved by losing your outgoing modifications. If you try to revert a resource not under version control, the resource will be deleted from the file system.

 **Note:** By default, a directory will be recursively reverted (including any other modified item it contains). However, if the directory has only property changes, you need to explicitly choose if the operation will include any modified items found inside it.

If you want some of your outgoing changes to be overridden you must first open the file in **Compare view** and choose the sections to be replaced with ones from the repository file. This can be achieved either by editing directly the file or by using the action **Copy change from right to left** from the **Compare view toolbar**. After editing the conflicting file you have to run the action **Mark as merged** before committing it.

If you want to drop all local changes and, at the same time, bring all incoming changes into your working copy resource, you can use the **Override and update** action which discards the changes in the local file and updates it from the repository. A dialog will show you the files that will be affected.



**Figure 362: Override and update dialog**

In the first table of the dialog you will be able to see the resources that will be overridden. In the second table you will find the list of resources that will be updated. Only resources that have an incoming status are updated.

 **Tip:** If you want to roll-back out of your working copy changes that have already been committed to the repository, see [Merge Revisions](#).

#### Merge Conflicted Resources

Before you can safely commit your changes to the repository you must first resolve all conflicts. In the case of pseudo-conflicts they can be resolved in most cases with an update operation which will merge the incoming modifications into your working copy resource. In the case of real conflicts, conflicts that persist after an update operation, it is necessary

to resolve the conflict using the built-in compare view and editor or, in the case of properties conflict, the **Properties view**. Before you can commit you must *mark as resolved* the affected files.

Both pseudo and real conflicts can be resolved without an update. You should open the file in the compare editor and decide which incoming changes need to be copied locally and which outgoing changes must be overridden or modified. After saving your local file you have to use the *Mark as merged* action from the contextual menu before committing.

### *Drop Incoming Modifications*

In the situation when your file is in conflict but you decide that your working copy file and its content is the correct one, you can decide to drop some or all of the incoming changes and commit afterwards. The action **Mark as merged** proves to be useful in this case too. After opening the conflicting files with **Compare view, Editor** or editing their properties in the **Properties** view and deciding that your file can be committed in the repository replacing the existing one, you should use the **Mark as merged** action. When you want to override completely the remote file with the local file you should run the action **Override and commit** which drops any remote changes and commits your file.

In general it is much safer to analyze all incoming and outgoing changes using the **Compare** view and only after to update and commit.

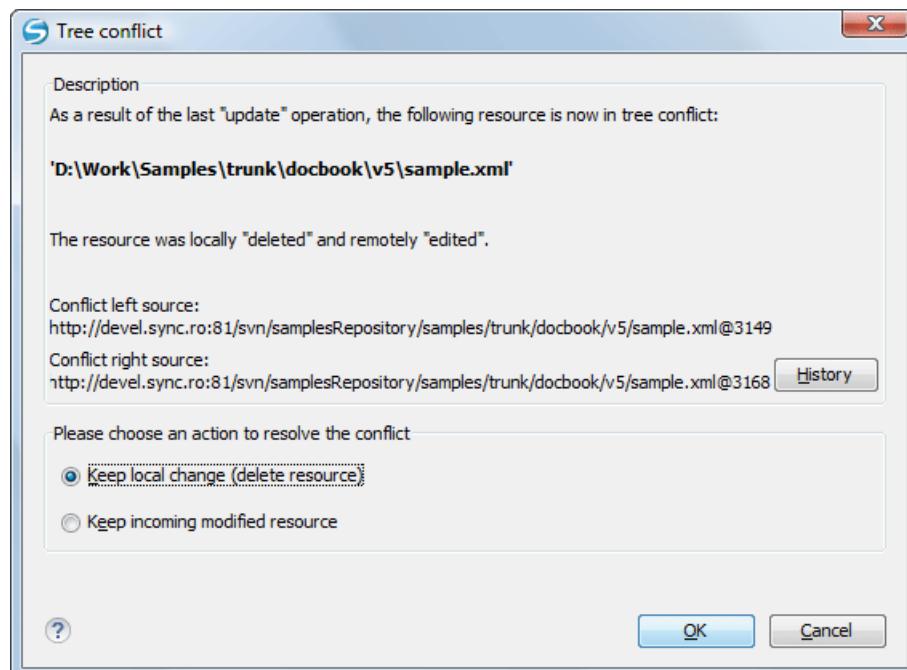
### *Tree Conflicts*

A *tree conflict* is a conflict at the directory tree structure level and occurs when the user runs an update action on a resource that:

- it is locally modified and the same resource was deleted from the repository (or deleted as a result of being renamed or moved);
- it was locally deleted (or deleted as a result of being renamed or moved) and the same resource is incoming as modified from the repository.

The same conflict situation can occur after a merge or a switch action. The action ends with an error and the folder containing the file that is now in the tree conflict state is also marked with a conflict icon.

Such a conflict can be resolved in one of the following ways which are available when the user double clicks on the conflicting resource or when running the **Edit conflict** action:



**Figure 363: Resolve a tree conflict**

- Keep the local modified file - If there is a renamed version of the file committed by other user that will be added to the working copy too.

- Delete the local modified file - Keeps the incoming change that comes from the repository.

## Update the Working Copy

While you are working on a project, other members of your team may be committing changes to the project repository. To get these changes, you have to *update* your working copy. Updating may be done on single files, a set of selected files, or recursively on entire directory hierarchies. The update operation can be performed from [Working Copy view](#). It updates the selected resources to the last synchronized revision (if remote information is available) or to the *HEAD* revision of the repository.

There are three different kinds of incoming changes:

- *Non-conflicting* - A non-conflicting change occurs when a file has been changed remotely but has not been modified locally.
- *Conflicting, but auto-mergeable* - An auto-mergeable conflicting change occurs when a text file has been changed both remotely and locally (i.e. has non-committed local changes) but the changes are on different lines of text. Not applicable to binary resources (for example multimedia files, PDFs, executable program files)
- *Conflicting* - A conflicting change occurs when one or more of the same lines of a text file have been changed both remotely and locally.

If the resource contains only incoming changes or the outgoing changes do not intersect with incoming ones then the update will end normally and the Subversion system will merge incoming changes into the local file. In the case of a conflicting situation the update will have as result a file with conflict status.

The Oxygen XML Editor allows you to update your working copy files to a specific revision, not only the most recent one. This can be done by using the **Update to revision/depth** action from the **Working Copy** view (**All Files** view mode) or the **Update to revision** action from the [History view](#) contextual menu.

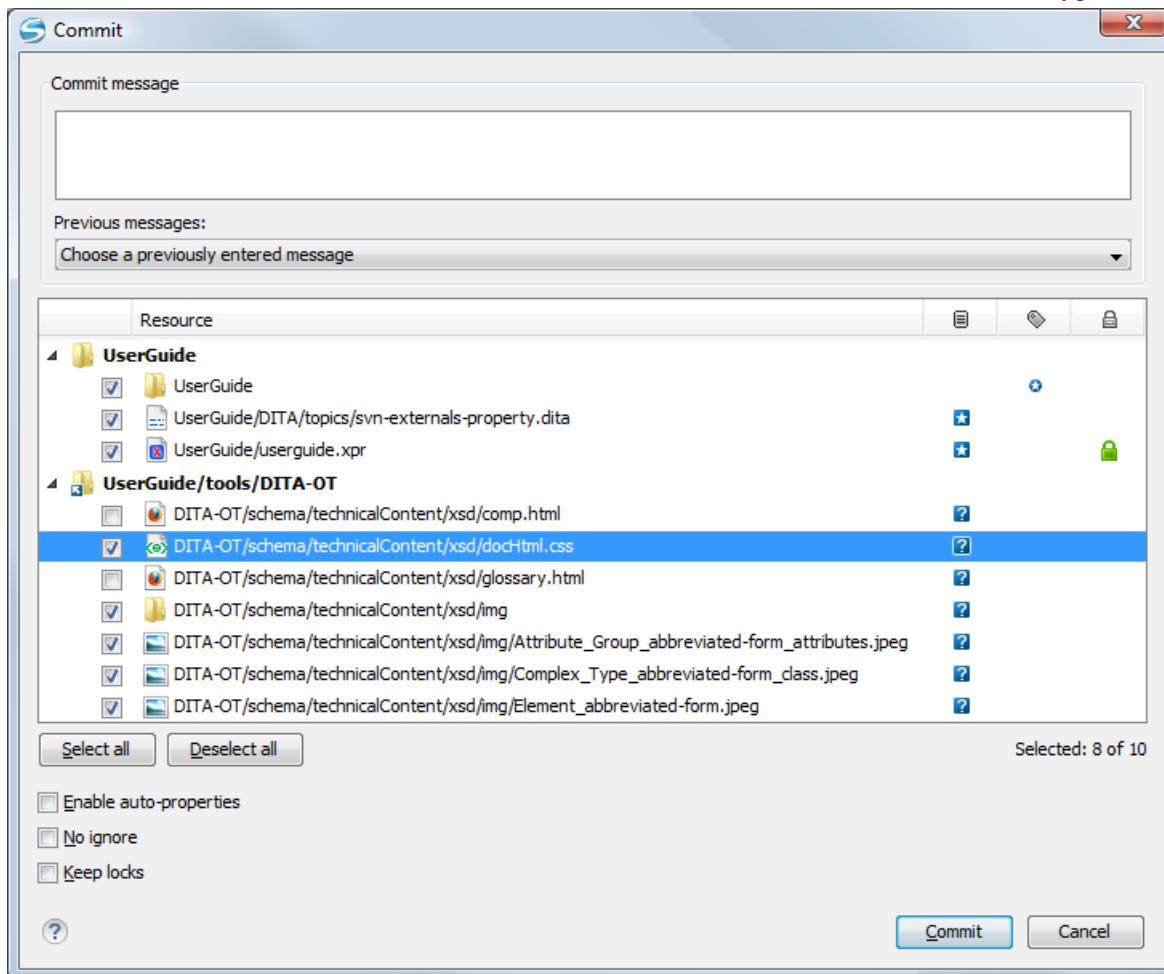
If you select multiple files and folders and then you perform an **Update** operation, all of those files and folders are updated one by one. The Subversion client makes sure that all files and folders belonging to the same repository are updated to the exact same revision, even if between those updates another commit occurred.

When the update fails with a message saying that there is already a local file with the same name Subversion tried to check out a newly versioned file, and found that an unversioned file with the same name already exists in your working folder. Subversion will never overwrite an unversioned file unless you specifically do this with an **Override and update** action. If you get this error message, the solution is simply to rename the local unversioned file. After completing the update, you can check whether the renamed file is still needed.

## Send Your Changes to the Repository

Sending the changes you made to your working copy is known as *committing* the changes. If your working copy is up-to-date and there are no conflicts, you are ready to commit your changes.

The **Commit** action sends the changes from your local working copy to the repository. The **Commit** dialog box presents all the items that you are able to commit.



**Figure 364: Commit dialog box**

Enter a message to associate with the commit, or choose a previous message from the **Previous messages** list (the last 10 commit messages will be remembered even after restarting the SVN client application).

An item that can be committed has one of the following states: *added*, *modified* (content or properties), *replaced*, and *deleted*. All items that have one of these states are selected in the dialog box by default. If you do not want to commit one of the items, uncheck it.

**Attention:** For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

Besides the items that have one of the mentioned states, Syncro SVN Client also includes the files being *unversioned* or *missing*. In order to be committed, these items are handled automatically:

- *Unversioned* items are added under version control.
- *Missing* items are deleted.

**Note:** In case the **Show unversioned directories content** is disabled, the **Commit** dialog box does not display the items inside an *unversioned* directory.

*Unversioned* or *missing* items are not selected by default in the **Commit** dialog box, unless you have selected them explicitly when issuing the commit command.



**Note:** In some cases, items that have one of the above states are not presented in the **Commit** dialog.

For example:

- Items that have been *added* or *replaced* previously, but now are presented as *missing* after being removed from the file system, outside of an SVN client. Such items do not exist in the repository and you should use the **Delete** action to remove them from your working copy.
- Items that have incoming changes from the repository, after a synchronization. You need to have your working copy up-to-date before committing your changes.
- Files that, after a synchronization, appear as locked by other users or from other locations than the current working copy.



**Note:** Due to dependencies between items, when you select or clear an *unversioned* (  ) or *added* (  ) item in the **Commit** dialog box, other items with one of these states can be selected or cleared automatically.

The modifications that will be committed for each file can be reviewed in the compare editor window by double clicking a file in the **Commit** dialog box, or by right clicking and selecting the **Show Modifications** action from the contextual menu. This option is available to review only file content changes, not property changes.

The  **Local file status** column indicates the actual state of the items and the  **Local properties status** column indicates whether the properties of an item are modified.

The  **Lock information** column is displayed in case at least one of the files in the **Commit** dialog box has lock information associated with it, valid against the commit operation.

The following options are available in this dialog box:

- **Enable automatic properties** or **Disable automatic properties** - enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in the `config` file of the Subversion configuration directory.  
 **Note:** This option is available only when there are defined properties to be applied automatically for resources newly added under version control. You can define these properties in the `config` file of the Subversion configuration directory, in the `auto-props` section. Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable automatic properties**, or **Disable automatic properties**.
- **Keep locks** - selecting the **Keep locks** option preserves any locks you set on various files.  
 **Note:** This option is available only when files that you locked are presented in the dialog box.

Each of the above options is activated only when you select an item for which the option can be applied.

Your working copy must be up-to-date with respect to the resources you commit. This is ensured by using the **Update** action prior to committing, resolving conflicts and re-testing as needed. If your working copy resources you are trying to commit are out of date you will get an appropriate error message.

### *Committing to Multiple Locations*

Although Subversion does not support committing to different locations at once, Syncro SVN Client offers this functionality regarding *external* items.

If items to be committed belong to different *external* definitions found in the working copy, they are grouped under the corresponding item that indicates their repository origin. Each parent item is rendered bold and its corresponding repository location is presented when hovering it. Parent items are decorated with a small arrow (  ) if they are *external* definitions. The working copy root directory is never decorated and is not presented if there are no *external* items listed (all items belong to the main working copy). Each child item is presented relative to the parent item.



**Note:** When an *external* directory has modifications of its own, it is presented both as a parent item and as an item that you can select and commit. This is always the case for *external* files.

The sets of items belonging to *external* definitions from the same repository are committed together, resulting a single revision. So, the number of revisions can be smaller than the number of *externals*. External definitions are considered from the same repository if they have the same protocol, server address, port, and repository address within the server.



**Note:** *External* files are always from the same repository as the parent directory which defines them, so they are always committed together with the changes from their parent directory.

## Integration with Bug Tracking Tools

Users of bug tracking systems can associate the changes they make in the repository resources with a specific ID in their bug tracking system. The only requirement is that the user includes the bug ID in the commit message that he enters in the **Commit** dialog. The format and the location of the ID in the commit message are configured with SVN properties.

To make the integration possible Syncro SVN Client needs some data about the bug tracking tool used in the project. You can configure this using the following *SVN properties* which must be set on the folder containing resources associated with the bug tracking system. Usually they are set recursively on the root folder of the working copy.

- **bugtraq:message** - A string property. If it is set *the Commit dialog* will display a text field for entering the bug ID. It must contain the string `%BUGID%`, which is replaced with the bug number on commit.
- **bugtraq:label** - A string property that sets the label for the text field configured with the **bugtraq:message** property.
- **bugtraq:url** - A string property that is the URL pointing to the bug tracking tool. The URL string should contain the substring `%BUGID%` which Syncro SVN Client replaces with the issue number. That way the resulting URL will point directly to the correct issue.
- **bugtraq:warnifnoissue** - A boolean property with the values *true/yes* or *false/no*. If set to *true*, the Syncro SVN Client will warn you if the bug ID text field is left empty. The warning will not block the commit, only give you a chance to enter an issue number.
- **bugtraq:number** - A boolean property with the value *true* or *false*. If this property is set to *false*, then any character can be entered in the bug ID text field. If the property is set to *true* or is missing then only numbers are allowed as the bug ID.
- **bugtraq:append** - A boolean property. If set to *false*, then the bug ID is inserted at the beginning of the commit message. If *yes* or not set, then it's appended to the commit message.
- **bugtraq:logregex** - This property contains one or two regular expressions, separated by a newline. If only one expression is set, then the bug ID's must be matched in the groups of the regular expression string, for example `[Ii]ssue #?(\\d+)` If two expressions are set, then the first expression is used to find a string which relates to a bug ID but may contain more than just the bug ID (e.g. `Issue #123` or `resolves issue 123`). The second expression is then used to extract the bug ID from the string extracted with the first expression. An example: if you want to catch every pattern `issue #XXX` and `issue #890, #789` inside a log message you could use the following strings:
  - `[Ii]ssue #?(\\d+)(,? ?#?(\\d+))+`
  - `(\\d+)`

The data configured with these SVN properties is stored on the repository when a revision is committed. A bug tracking system or a statistics tools can retrieve from the SVN server the revisions that affected a bug and present the commits related to that bug to the user of the bug tracking system.

If the **bugtraq:url** property was filled in with the URL of the bug tracking system and this URL includes the `%BUGID%` substring as specified above in the description of the **bugtraq:url** property then *the History view* presents the bug ID as a hyperlink in the commit message. A click on such a hyperlink in the commit message of a revision opens a Web browser at the page corresponding to the bug affected by that commit.

## Obtain Information for a Resource

This section explains how to obtain information for a SVN resource:

### Request Status Information for a Resource

While you are working with the SVN Client you often need to know which files you have changed, added, removed, or renamed, or even which files got changed and committed by others. This is where the **Synchronize** action from the

**Working Copy view** comes in handy. The **Working Copy** view shows you every file that has changed your working copy, as well as any unversioned files you may have.

If you want more detailed information about a given resource, you can use the  **Show SVN Information** action. This action is available from the **File** menu or the contextual menu of the **Working Copy**, **Repositories**, **History**, or **Directory Change Set** views, or from the **Revision Graph** dialog box. The **SVN Information** dialog box will be displayed, showing information about the selected resource. The information displayed depends on the location of the item (local or remote) and may include the following:

- Local path and repository location
- Revision number
- Last change author, revision and date
- Information about locks
- Local file status
- Local properties status
- Local directory depth
- Repository location and revision number for copied files or directories
- Path information about locally moved items
- Path information about conflict generated files
- Remote file status
- Remote properties status
- File size and other information

The value of a property of the resource displayed in the dialog box can be copied by right-clicking on the property and selecting the **Copy** action.

### Request History for a Resource

In Apache Subversion™, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from **Repositories view**, **Working Copy view**, **Revision Graph**, or **Directory Change Set view**. From the **Working copy view** you can display the history of local versioned resources.

### Management of SVN Properties

In the **Properties view** you can read and set the Apache Subversion™ properties of a file or folder. There is a set of predefined properties with special meaning to Subversion. For more information about properties in Subversion see the SVN Subversion specification. Subversion properties are revision dependent. After you change, add or delete a property for a resource, you have to commit your changes to the repository.

If you want to change the properties of a given resource you need to select that resource from the **Working Copy view** and run the **Show properties** action from the contextual menu. The **Properties** view will show the local properties for the resource in the working copy. Once the **Properties** view is visible, it will always present the properties of the currently selected resource. In the **Properties** view **toolbar** there are available actions which allow you to add, change and delete the properties.

If you choose the **Add a new property** action, a new dialog will pop-up containing:

- **Name** - Combo box which allows you to enter the name of the property. The drop down list of the combo box presents the predefined Subversion properties such as **svn:ignore**, **svn:externals**, **svn:needs-lock**, etc.
- **Current value** - Text area which allows you to enter the value of the new property.

If the selected item is a directory, you can also set the property recursively on its children by checking the **Set property recursively** checkbox.

If you want to change the value for a previously set property you can use the **Edit property** action which will display a dialog where you can set:

- **Name** - Property name (cannot be changed).
- **Current value** - Presents the current value and allows you to change it.

- **Base value** - The value of the property, if any, from the resource in the pristine copy. It cannot be modified.

If you want to completely remove a property previously set you can choose the **Remove property** action. It will display a confirmation dialog in which you can choose also if the property will be removed recursively.

In the [Properties view](#) there is a **Refresh** action which can be used when the properties have been changed from outside the view. This can happen, for example, when the view was already presenting the properties of a resource and they have been changed after an **Update** operation.

## Branches and Tags

One of the fundamental features of version control systems is the ability to create a new line of development from the main one. This new line of development will always share a common history with the main line if you look far enough back in time. This line is known as a *branch*. Branches are mostly used to try out features or fixes. When the feature or fix is finished, the branch can be merged back into the main branch (*trunk*).

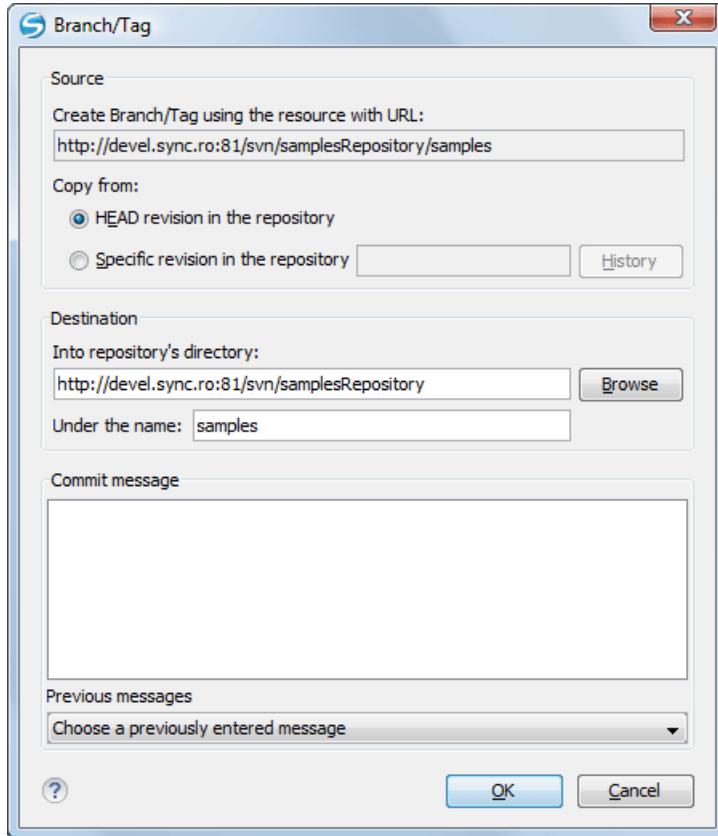
Another feature of version control systems is the ability to take a snapshot of a particular revision, so you can at any time recreate a certain build or environment. This is known as *tagging*. Tagging is especially useful when making release versions.

In Apache Subversion™, there is no difference between a *tag* and a *branch*. On the repository, both are ordinary directories that are created by copying. The trick is that they are cheap copies instead of physical copies. Cheap copies are similar to hard links in Unix, which means that they merely link to a specific tree and revision without making a physical copy. As a result, branches and tags occupy little space on the repository and are created very quickly.

As long as nobody ever commits to the directory in question, it remains a tag. If people start committing to it, it becomes a branch.

### Create a Branch / Tag

To create a branch or tag by copying a directory, use the **Branch/Tag...** action that is available in the **Tools** menu when an item is selected in the [Working Copy view](#) or [Repositories view](#), or from the contextual menu of the **Repositories** view.



**Figure 365: The Branch/Tag Dialog Box**

You can configure the following options in this dialog box:

You can specify the source revision of the copy in the **Copy from** section. You can choose between the following options:

- **HEAD revision in the repository** - The new branch or tag will be copied in the repository from the HEAD revision. The branch will be created very quickly, as the repository will make a *cheap* copy.
- **Specific revision in the repository** - The new branch will be copied into the repository, but you can specify the exact desired revision. For example, this is useful if you forgot to make a branch or tag when you released your application. If you click the **History** button you can select the revision number from *the History dialog box*. This type of branch will also be created very quickly.
- **Working copy** - (Available only if the item is selected from the **Working copy** view). The new branch will be a copy of your local working copy. If you have updated some files to an older revision in your working copy, or if you have made local changes, that is exactly what goes into the copy. This involves transferring some data from your working copy back to the repository, or more specifically, the locally modified files.

You can specify the location of the new branch or tag in the **Destination** section:

- **Into repository's directory** - *The URL of the parent directory* of the new branch or tag.
-  **Note:** *Peg revisions* have no effect for this operation since it is used to send information to the repository.
- **Under the name** - You can specify another branch or tag name other than the name of the resource selected in the **Repositories** or **Working copy** view.

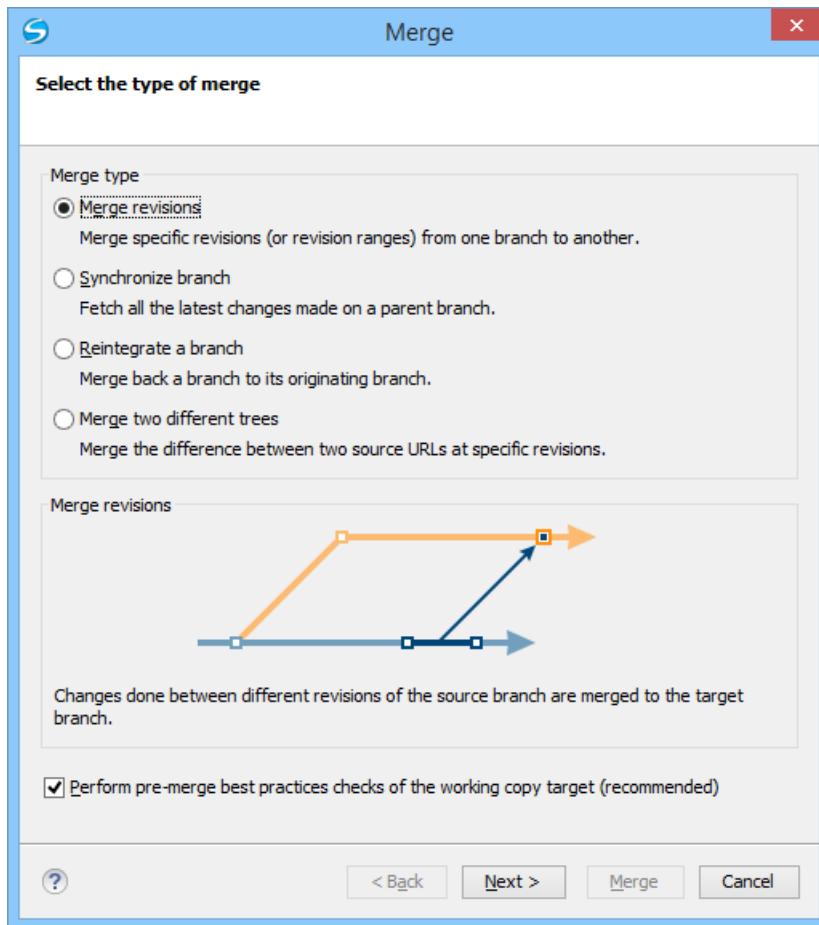
The new branch or tag will be created as a child of the specified URL of the repository directory and will have the new name.

## Merging

At some stage during the development process, you will want to merge the changes made on a *branch* back into the *trunk*, or vice-versa. The *merge* is accomplished by comparing two points (branches or revisions) in the repository and applying the obtained differences to your working copy. This process is closely related to the *diff* concept.

 **Note:** A *branch* is a line of development that exists independently of another line, yet still shares a common history if you look far enough back in time. A *branch* always begins life as a *copy of something* (such as a *trunk*, another *branch*, or *tag*), and moves on from there, generating its own history.

The  **Merge...** action is available in the **Tools** menu. The working copy item selected when you issued the command will be the one receiving the generated changes. If there is no item selected, the *merge* operation will be performed on the entire working copy.



**Figure 366: The Merge Wizard**

The four types of merging are as follows:

- **Merge revisions** - port changes from one branch to another. Note that the *trunk* can also be considered a branch, in this context.
- **Synchronize branch** - fetch all the changes made on a parent branch (or the *trunk*) to a child branch.
- **Reintegrate a branch** - merge back a branch to its parent branch (which can also be the *trunk*).
- **Merge two different trees** - integrate the changes done on a branch to a different branch.

It is recommended that you enable the following pre-merge check option:

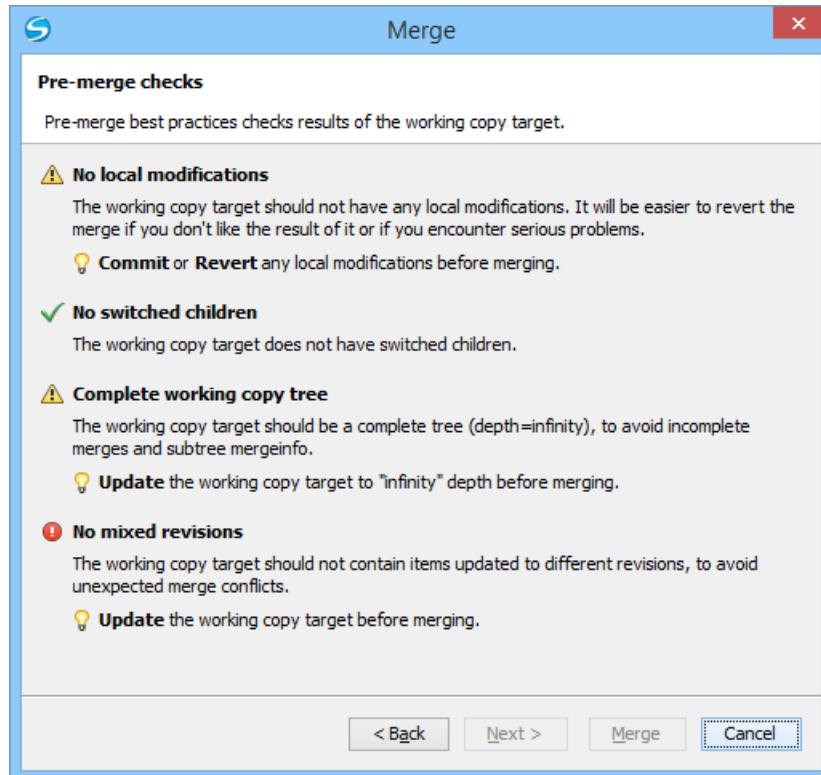
- **Perform pre-merge best practices checks of the working copy target** - When enabled, the SVN Client checks if the working copy target item is ready for the merge operation and displays the **pre-merge checks** wizard page.

**R** **Remember:** It is a good idea to perform a merge into an unmodified working copy. If you have made changes to your working copy, commit them first. If the *merge* does not go as you expect, you may want to revert the changes and revert cannot recover your uncommitted modifications.

**!** **Important:** The above recommendation becomes mandatory when *reintegrating a branch*.

### Pre-Merge Checks

Before performing a merge, it is recommended to make sure that the working copy target item is ready for the merge operation. The SVN Client includes a best practices step that checks various conditions of the working copy target item to ensure that the merge operation will succeed. By enabling the **Perform pre-merge best practices checks of the working copy target** option in the first page of the **Merge** wizard, the **Pre-merge checks** wizard page is displayed to give you a summary of the verified conditions.



**Figure 367: The Pre-Merge Checks Wizard Page**

The following conditions are checked in this operation:

#### No local modifications

The working copy item (or any of its children) receiving the merge should not contain uncommitted changes, to make it easier to revert merge-generated changes if you encounter unexpected results.

**i Tip:** If this condition fails, you should *commit* or *revert* the local modifications before merging.

#### No switched children

None of the children of the working copy item receiving the merge should be switched, to avoid incomplete merges and *subtree mergeinfo*.

**i Tip:** If this condition fails, you should switch back all the children before merging.

#### Complete working copy tree

The working copy item receiving the merge should be a complete directory tree structure with an infinite depth, to avoid incomplete merges and *subtree mergeinfo*.

-  **Tip:** If this condition fails, you should change the *sticky* depth of the working copy item receiving the merge to *infinity* value.

## No mixed revisions

The working copy item receiving the merge should not contain items that were updated to different revisions, to avoid unexpected merge conflicts.

-  **Tip:** If this condition fails, you should *update* the working copy before merging.

Each condition is marked with an icon that represents the state of the condition. The possible states are as follows:

-  **Successful** - The condition is fulfilled successfully.
-  **Warning** - The condition is not fulfilled, but it is not mandatory.
-  **Error** - The condition is not fulfilled and is mandatory, and therefore the operation cannot proceed until you solve the error.

-  **Tip:** For each condition state, a message is displayed that gives you additional information about the results and, for warning or errors, a hint that explains how you can solve them.

-  **Important:** After solving any of the warnings or errors, it is recommended that you perform the *pre-merge checks* again to make sure your new changes are valid.

## Merge Revisions

This is the case when you have made one or more changes to a branch and you want to duplicate them in a different branch. For example, we know that a problem has been fixed by committing revisions 17, 20, and 25 on branch B1. These changes are also needed in branch B2. Thus, in order to merge them, we need a working copy of the B2 branch.

To merge revisions from a different branch, follow these steps:

1. Go to menu **Tools > Merge**.  
The **Merge** wizard is opened.
2. Select the **Merge revisions** option.
3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
  - a) Press the **Next** button.  
If the **Perform pre-merge best practices checks of the working copy target** option was enabled, [the Pre-Merge Checks wizard page](#) is displayed.



**Note:** If errors are found you need to solve them before proceeding.

4. Press the **Next** button.  
The **Merge revisions** wizard page is displayed.
5. In the **Merge from (URL)** text box, enter [the URL of the branch or tag](#) that contain the changes that you want to duplicate in your working copy. In our example, it is the URL of the B1 branch.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have previously merged from this branch, then you can simply use the drop down list, which shows a history of previously used URLs.



**Note:** If the URL belongs to a different repository than the working copy, the **Ignore ancestry / Disable merge tracking** option, in the [Merge Options wizard page](#), will be enabled automatically (and you cannot change this). This is because the [Subversion client cannot track changes between different repositories](#).



**Tip:** You can also specify a [peg revision](#) at the end of the URL (for example, URL@rev1234). The peg revision does not affect the merge range you select. By default, the HEAD revision is assumed.

6. In the **Revisions to merge** section, choose between the **all revisions** and **specific revision(s)** options.

- **all revisions** - The operation will include *all eligible revisions* that were not yet merged.
- **specific revision(s)** - You can specify one or more individual revisions and/or revision ranges. Also, you can mix *forward* ranges (for example, 1–5), *backward* ranges (for example, 20–15), and subtract specific revisions from a range (for example, 1–5, –3).

 **Note:** If using the Subversion command-line client, a revision range of the form 1–5 means all changes starting from revision 2 up to revision 5 (the changes necessary to reach revision 5, committed after revision 1). Unlike the Subversion command-line client, in Syncro SVN Client the revision ranges are inclusive, meaning that it will process all revisions, starting with revision 1, up to and including revision 5.

 **Attention:** The **HEAD** revision is the only non-numerical revision allowed, and it can only be used when specifying revision ranges as one of the ends of the range (for example, 10–**HEAD**). Be careful when using it, as it might not refer to the desired revision, if it has recently been committed by another user.

 **Tip:** If you want to perform a *reverse merge* and roll-back your working copy changes that have already been committed to the repository, use the *negative revisions* notation (for example, –7) or *backward revision ranges* (for example, 20–10).

- a) If you press the **History** button, *the History dialog box* is displayed, which allows you to select one or more revisions to be merged.

7. Optionally, if you want to *configure the options* for your merge, press the **Next** button.

The *Merge Options wizard page* is displayed that allows you to configure options for the operation.

 **Warning:** If the **Ignore ancestry / Disable merge tracking** option is enabled and you selected **all revisions** in the **Revisions to merge** section, revisions that were previously merged will also be included, which may result in conflicts.

8. Press the **Merge** button.

The merge operation is performed.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

 **Note:** The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

### Synchronize a Branch

While working on your own branch, other people on your team might continue to make important changes in the parent branch (which can be the *trunk* itself or any other branch). It is recommended to periodically duplicate those changes in your branch to make sure your changes are compatible with them. This is done by performing a *synchronize merge*, which will bring your branch up-to-date with any changes made to its ancestral parent branch since the time your branch was created or last synchronized. Subversion is aware of the history of your branch and can detect when it split away from the parent branch.

Frequently keeping your branch in sync with the parent branch helps you to prevent unexpected conflicts when the time comes for you to duplicate your changes back into the parent branch. The synchronization uses *merge tracking* to skip all those revisions that have already been merged, thus a sync merge can be repeated periodically to fetch all the latest changes of the parent branch to keep up-to-date with it.

 **Important:** It is recommended to synchronize the whole working copy that was created from the child branch (the root of the working copy), rather than just a part of it.

After running the *synchronize merge*, your working copy from the child branch now contains new local modifications, and these edits are duplications of all of the changes that have happened on the *trunk* since you first created your branch. At this point, your private branch is now synchronized with the trunk.

To synchronize your branch with its parent branch, follow these steps:

1. Go to **Tools > Merge**.  
The **Merge** wizard is opened.
2. Select the **Synchronize branch** option.
3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
  - a) Press the **Next** button.  
If the **Perform pre-merge best practices checks of the working copy target** option was enabled, [the Pre-Merge Checks wizard page](#) is displayed.
 

 **Note:** If errors are found you need to solve them before proceeding.
4. Press the **Next** button.  
The **Synchronize branch** wizard page is displayed.
5. In the **Parent branch (URL)** text box, enter [the URL of the branch from which you created your branch](#). This means that the URL must belong to the same repository as your working copy that was created from the child branch.  
You may also click the **Browse** button to browse the repository and find the desired branch. If you have previously merged from this branch, then you can simply use the drop down list, which shows a history of previously used URLs.
 

 **Tip:** You can also specify a [peg revision](#) at the end of the URL (for example, URL@rev1234). The peg revision specifies both the peg revision of the URL and the latest revision that will be considered for merging. By default, the HEAD revision is assumed.
6. Optionally, if you want to [configure the options](#) for your merge, press the **Next** button.  
The [Merge Options wizard page](#) is displayed that allows you to configure options for the operation.
 

 **Note:** The **Ignore ancestry / Disable merge tracking** option is not available for this merge type, since a synchronization merge should always be recorded in the destination branch.
7. Press the **Merge** button.  
The merge operation is performed.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, [you may need to resolve conflicts](#) after making major changes.

 **Note:** The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

### Reintegrate a Branch

There are some conditions that apply to reintegrate a branch:

- The server must support merge tracking.
- The source branch (to be reintegrated) must be coherently synchronized with its parent branch. This means that all revisions between the branching point and the last revision merged from the parent branch to the child branch must be merged to the latter one (there must be no missing revisions in-between).
- The working copy **must not**:
  - have any local modifications.

- contain a mixture of revisions (all items must point to the same revision).
- have any sparse directories (all directories must be of depth *infinity*).
- contain any switched items.
- The revision of the working copy must be greater than or equal to the last revision of the parent branch with which the child branch was synchronized.

 **Tip:** You can use [the pre-merge checks option](#) to make sure these conditions are fulfilled.

This method is useful when you have a feature branch on which the development has concluded and it should be merged back into its parent branch. Since you have kept the feature branch synchronized with its parent, the latest versions of them will be absolutely identical except for your feature branch changes. These changes can be reintegrated into the parent branch by using a working copy of it and the **Reintegrate a branch** option.

This method uses the *merge-tracking* features of Apache Subversion™ to automatically calculate the correct revision ranges and to perform additional checks that will ensure that the branch to be reintegrated has been fully updated with its parent changes. This ensures that you do not accidentally undo work that others have committed to the parent branch since the last time you synchronized the child branch with it. After the merge, all branch development will be completely merged back into the parent branch, and the child branch will be redundant and can be deleted from the repository.

 **Tip:** Before reintegrating the child branch it is recommended to synchronize it with its parent branch one more time, to help avoid any possible conflicts.

To reintegrate a child branch into its parent branch, follow these steps:

1. Go to menu **Tools > Merge**.  
The **Merge** wizard is opened.
2. Select the **Reintegrate a branch** option.

 **Note:** This option is disabled if the selected working copy item (or if it is a directory, any of the items inside of it) has any type of modification. This is because it is mandatory for the target item to have no modifications.

3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
  - a) Press the **Next** button.  
If the **Perform pre-merge best practices checks of the working copy target** option was enabled, [the Pre-Merge Checks wizard page](#) is displayed.

 **Note:** If errors are found you need to solve them before proceeding.

4. Press the **Next** button.  
The **Reintegrate a branch** wizard page is displayed.
5. In the **Child branch (URL)** text box, enter [the URL of the child branch to be reintegrated](#). This means that the URL must belong to the same repository as your working copy that was created from the parent branch.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have previously merged from this branch, then you can simply use the drop down list, which shows a history of previously used URLs.

 **Tip:** You can also specify a [peg revision](#) at the end of the URL (for example, URL@rev1234). The peg revision specifies both the peg revision of the URL and the latest revision that will be considered for merging. By default, the HEAD revision is assumed.

6. The [Merge Options wizard page](#) is displayed that allows you to configure options for the operation.

 **Note:** Because a *reintegrate merge* is so specialized, most of the merge options are not available, except for those in the **File Comparison** category.

7. Press the **Merge** button.

The merge operation is performed.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

-  **Note:** The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

### Merge Two Different Trees

This merge type is useful when you need to duplicate changes from one child branch (for example, CB1) to another child branch (CB2) from the same parent branch. The SVN client will calculate the changes necessary to get from the HEAD revision of the parent branch (or the *trunk*) to the HEAD revision of one of its child branches (CB1), and apply those changes to your working copy of the other branch (CB2). The result is that the latter child branch (CB2) will also include the changes made on the original child branch (CB1), although that branch was not reintegrated into the parent branch.

This merge type could also be used to reintegrate a child branch back into its parent when the repository does not support *merge tracking*.

-  **Note:** If the server does not support *merge-tracking*, then this is the only way to merge a branch back to its parent.

1. Go to menu **Tools > Merge**.  
The **Merge** wizard is opened.
2. Select the option **Merge two different trees**.
3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
  - a) Press the **Next** button.  
If the **Perform pre-merge best practices checks of the working copy target** option was enabled, *the Pre-Merge Checks wizard page* is displayed.

-  **Note:** If errors are found you need to solve them before proceeding.

4. Press the **Next** button.  
The **Merge two different trees** wizard page is displayed.
5. In the **From (starting URL and revision)** section enter *the URL of the first branch*.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have previously merged from this branch, then you can simply use the drop down list, which shows a history of previously used URLs.

-  **Tip:** If you are using this method to merge a feature branch back to its parent branch, you need to start the merge wizard from within a working copy of the parent. In this field enter the full URL of the parent branch. This may sound wrong, but remember that the parent is the starting point to which you want to add the branch changes.

-  **Note:** If the URL belongs to a different repository than the working copy, the **Ignore ancestry / Disable merge tracking** option, in the *Merge Options wizard page*, will be enabled automatically (and you cannot change this). This is because the *Subversion client cannot track changes between different repositories*.

-  **Tip:** You can also specify a *peg revision* at the end of the URL (for example, URL@rev1234). By default, the HEAD revision is assumed.

6. Enter the last revision number at which the two trees were synchronized by choosing between **HEAD revision** and **other revision**.
  - **HEAD revision** - Use this option if you are sure that no one else has committed changes since the last synchronization.
  - **other revision** - Use this option to input a specific revision number and avoid losing recent commits. You can use the **History** button to see a list of all revisions.

7. In the **To (ending URL and revision)** section enter *the URL of the second branch*.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have previously merged from this branch, then you can simply use the drop down list, which shows a history of previously used URLs.

 **Tip:** If you are using this method to merge a feature branch back to its parent branch, enter the URL of the feature branch. This way, only the changes unique to this branch will be merged, since the branch should have been periodically synchronized with its parent.

 **Attention:** The URL must point to the same repository as the one in the **From (starting URL and revision)** field. Otherwise, the operation will not be allowed, since Subversion cannot compute changes between items from different repositories.

 **Tip:** You can also specify a *peg revision* at the end of the URL (for example, URL@rev1234). By default, the HEAD revision is assumed.

8. Select a revision to compute all changes committed up to that point by choosing between **HEAD revision** and **other revision**.

- **HEAD revision** - This is the default selected revision.
- **other revision** - Use this option if you want to enter a previous revision. You can use the **History** button to see a list of all revisions.

9. Optionally, if you want to *configure the options* for your merge, press the **Next** button.

The *Merge Options wizard page* is displayed that allows you to configure options for the operation.

 **Warning:** If the **Ignore ancestry / Disable merge tracking** option is enabled and you selected **all revisions** in the **Revisions to merge** section, revisions that were previously merged will also be included, which may result in conflicts.

10. Press the **Merge** button.

The merge operation is performed.

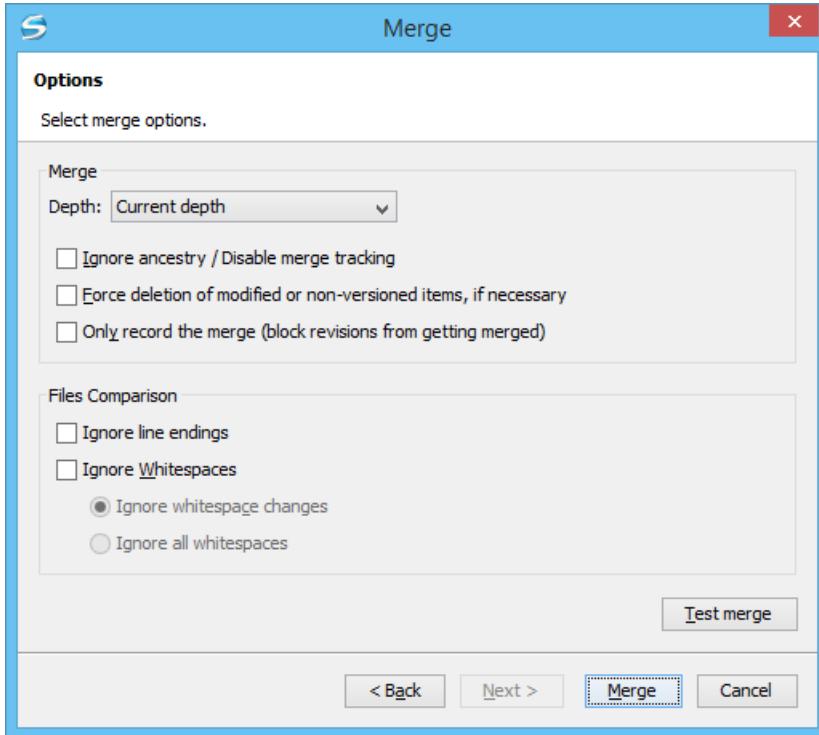
If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

 **Note:** The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

### *Merge Options*

Here is the list of options that can be used when merging:



**Figure 368: The Merge Wizard - Advanced Options**

- **Depth** (This option is applicable only for directories) - sets the depth of the merge operation. You can specify how far down into your working copy the merge should go by selecting one of the following values:
  - **Current depth** - Obeys the depths registered for the directories in the working copy that are to be switched.
  - **Recursive (infinity)** - Merges all the files and folders contained in the selected folder. This is the recommended depth for most users, to avoid incomplete merges and *subtree mergeinfo*.
  - **Immediate children (immediates)** - Merges only the child files and folders without recursing subfolders.
  - **File children only (files)** - Merges only the child files.
  - **This folder only (empty)** - Merges only the selected folder (no child files or folders are included).
- **Note:** The *depth* term is described in the [Sparse checkouts](#) section. The default depth is the current depth of the working copy item receiving the merge.
- **Ignore ancestry / Disable merge tracking** - Changes the way two items are merged if they do not share a common ancestry. Most merges involve comparing items that are ancestrally related to one another. However, occasionally you may want to merge unrelated items. If this option is disabled, the first item will be replaced with the second item. In these situations, you would want the merge to do a path-based comparison only, ignoring any relations between the items. For example, if two different files have the same name and are in the same relative location, disabling the option replaces one of the files with the other one, and enabling it merges their contents.
  - **Note:** If the URL of the merge source belongs to a different repository than the URL of the target working copy item (the one receiving the changes), this option is selected automatically (and you cannot change this). This is because the [Subversion client cannot track changes between different repositories](#).
- **Force deletion of modified or non-versioned items, if necessary** - If disabled, when the merge operation involves deleting locally modified or non-versioned items, it will fail. This is done in order to prevent data loss. This option is only available if there are uncommitted changes in the working copy.
- **Only record the merge (block revisions from getting merged)** - Available when the **Ignore ancestry / Disable merge tracking** option is disabled. It enables a special mode of the merge operation that just records it in the local merge tracking information, without actually performing it (does not modify any file contents or the structure of your working copy). You might want to enable this option for two possible reasons:

- You made (or will make) the merge manually, and therefore you need to mark the revisions as being merged to make the merge tracking system aware of them. This will exclude them from future merges.
- You want to prevent one or more particular changes from being fetched in subsequent merges.
- **Ignore line endings** - Allows you to specify how the line ending changes should be handled. By default, all such changes are treated as real content changes, but you can ignore them if you select this option.
- **Ignore whitespaces** - Allows you to specify how the whitespace changes should be handled. By default, all such changes are treated as real content changes, but you can ignore them if you select this option.
  - **Ignore whitespace changes** - Ignores changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces).

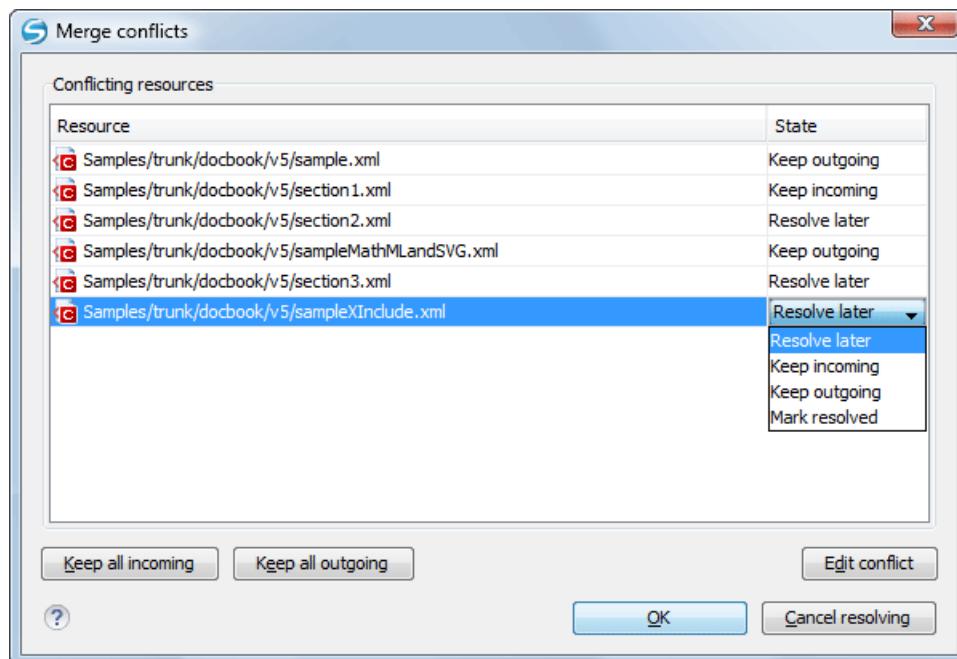


**Note:** Whitespaces that were added where there were none before, or that were removed, are still considered to be changes.

- **Ignore all whitespaces** - Ignores all types of whitespace changes.
- **Test merge** - Performs a dry run of the merge operation, allowing you to *preview* it without actually performing the merge. In the **Console** view you will see a list of the working copy items that will be affected and how they will be affected. This is helpful in detecting whether or not a merge will be successful, and where conflicts may occur.

### Resolving Merge Conflicts

After the merge operation is finished, it is possible to have some items in conflict. This means that some incoming modifications for an item could not be merged with the current working copy version. If there are such conflicts, the **Merge conflicts** dialog box will appear, presenting the items that are in conflict. This dialog box offers you choices for resolving the conflicts.



**Figure 369: Merge Conflicts Dialog**

The options to resolve a conflict are as follows:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications and discards all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy and discards all incoming ones.

- **Mark resolved** - You should choose this option after you have manually solved the conflict, to instruct the Subversion that it was resolved. To do this, use the **Edit conflict** button, which displays a dialog box that presents the contents of the conflicting items (the content of the working copy version versus the incoming version).

#### *Additional Notes About the Merge Operation*

##### **Sub-tree Merges**

It is recommended to perform a merge on the whole working copy (select its root directory when triggering the operation) to avoid *sub-tree mergeinfo*. *Sub-tree mergeinfo* is the *mergeinfo* recorded to describe a *sub-tree merge*. That is, a merge done directly to a child of a branch root that might be needed in certain situations. There is nothing special about *sub-tree merges* or *sub-tree mergeinfo* except that the complete record of merges to a branch may not be contained solely in the *mergeinfo* on the branch root and you may have to look to any *sub-tree mergeinfo* to get a full accounting. Fortunately, Subversion does this for you and rarely will you need to look for it.

##### **Merging from Foreign Repositories**

Subversion supports merging from foreign repositories. While all merge source URLs must point to the same repository, the merge target (from the working copy) may come from a different repository than the sources. However, copies made in the merge source will be transformed into plain additions in the merge target. Also, *merge-tracking* is not supported for merges from foreign repositories.

-  **Note:** When performing merges from repositories other than the one corresponding to the target item (from the working copy), the **Ignore ancestry / Disable merge tracking** option, in the [Merge Options wizard page](#), will be enabled automatically (and you cannot change this).

##### **General Merge Recommendations**

As a recommendation, you should only merge into clean working copies that **do not** contain any of the following:

- Modifications.
- Sparse directories (all directories must be of depth *infinity*).
- Switched items.

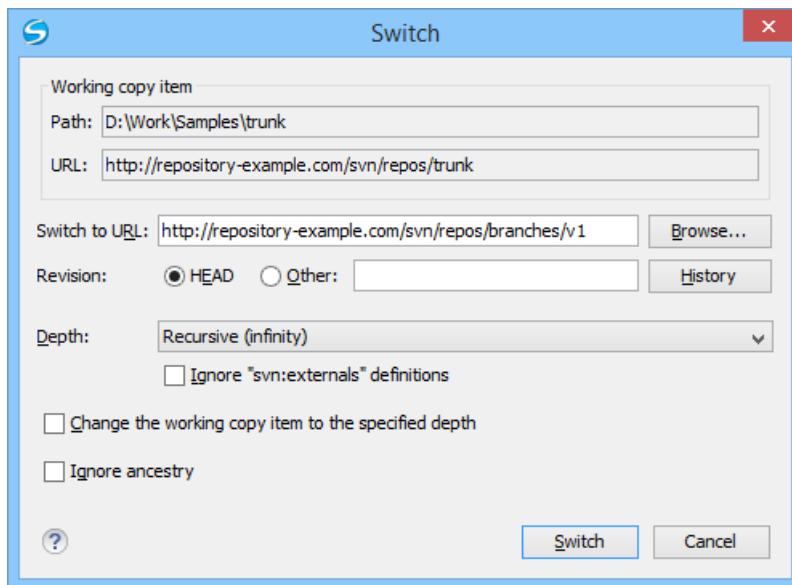
 **Important:** This recommendation becomes mandatory when performing a [reintegrate merge](#) operation. Also, trying to merge to mixed-revision working copies will fail in all types of merge operations.

 **Remember:** The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

##### **Switch the Repository Location**

The **Switch** action is useful when the repository location of a working copy, or an already committed item in the working copy, must be changed within the same repository. The action is available on the **Tools** menu when a versioned resource is selected in the current working copy that is displayed in [the Working Copy view](#).

 **Note:** *External* items cannot be switched using this action. Instead, change the value of the `svn:externals` property set on the parent directory of the external item and update the parent directory.



**Figure 370: The Switch Dialog Box**

The following options can be configured in the **Switch** dialog box:

#### Switch to URL

*The new location in the same repository* to which you are switching.



**Tip:** You can switch to items that were deleted, moved, or replaced, by specifying the original URL (before the item was removed) and use a *peg revision* at the end (for example, URL@rev1234).



**Note:** For items that are already *switched* that you want to switch back, the proposed URL is the original location of the item.

#### Revision

The specific version of the location to which you are switching.

#### Depth - (This option is applicable only for directories)

**Current depth** - Obeys the depths registered for the directories in the working copy that are to be switched.

**Recursive (infinity)** - Switches the location of the selected folder and all of its files and folders.

**Immediate children (immediates)** - Switches the location of the selected folder and its child files and folders without recursing subfolders.

**File children only (files)** - Switches the location of the selected folder and its child files.

**This folder only (empty)** - Switches the location of the selected folder (no child files or folders are included).

#### Ignore "svn:externals" definitions

When enabled, external items are ignored in the switch operation. This option is only available if you choose the **Current depth** or **Recursive (infinity)** depth.

#### Change the working copy item to the specified depth

Changes the *sticky* depth on the directory in the working copy.

#### Ignore ancestry

When disabled, the SVN system does not allow you to switch to a location that does not share a common ancestry with the current location. If enabled, the SVN does not check for a common ancestry.

#### Relocate a Working Copy

Sometimes the base URL of the repository is changed after a working copy is checked out from that URL. For example, if the repository itself is moved to a different server. In such cases, you do not have to check out a working copy from

the new repository location. It is easier to change the base URL of the root folder of the working copy to [the new URL of the repository](#).

 **Note:** *Peg revisions* have no effect for this operation.

This **Relocate** action is available in the **Tools** menu when selecting the root item of the working copy.

 **Note:** *External* items that are defined using absolute URLs and that point to the same repository as the working copy are not affected by the **Relocate** action (the URL is not updated). To relocate these items, change the value of the `svn:externals` property for each external item, which is set on their parent directories. For example, if an external item is defined as `externalDir http://host/path/to/repo/to/dir` and the repository was moved to another server (`host2`) and its protocol changed to `https`, then the value of the property needs to be updated to `externalDir https://host2/path/to/repo/to/dir`.

 **Tip:** If you edit external items using the method described in the previous note, on the next update the system will try to fetch the external items again. To avoid this, you can invoke the **Relocate** action on each of these external items.

## Patches

This section explains how to work with patches in Syncro SVN Client.

### What is a Patch

Suppose you are working with a set of XML files that you want to tag the project and distribute releases to other team members. While working on the project and correcting problems, you may need to distribute the corrections to other team members. In this case, you can distribute a patch, which is a collection of differences, that applied over the last distribution, would correct the problems. By default, the Syncro SVN Client generates patches in [the Unified Diff format](#), but it can also use [the Git format](#).

Creating a patch in Apache Subversion™ implies the access to either two revisions of a versioned item, or two different versioned items from the repository:

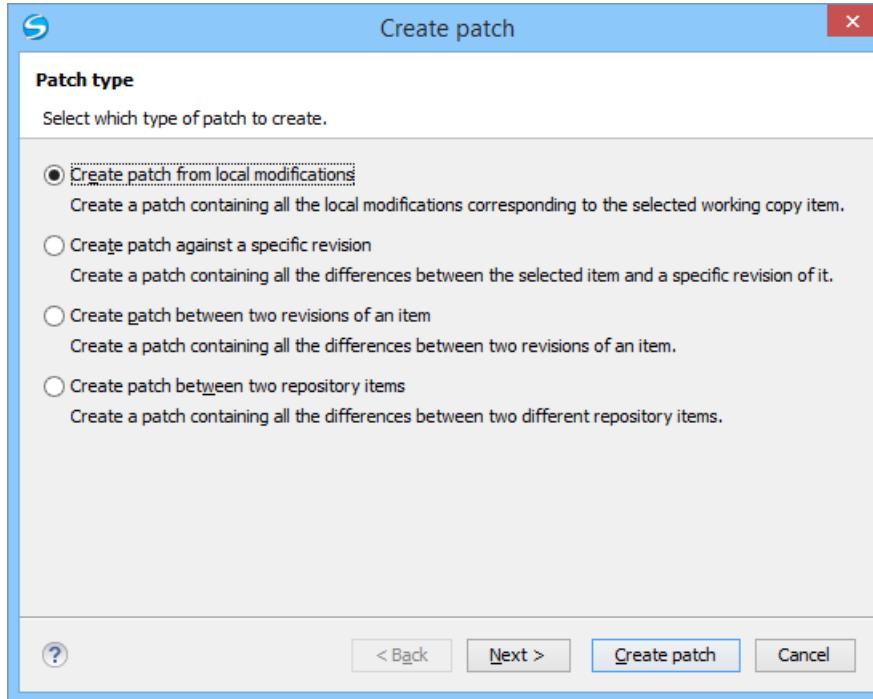
- *Two revisions of a version item* - the item can be local or remote and you can select two versions of it. This also applies when you need to generate a patch that only contains the changes in the working copy that were not yet committed.
- *Two different versioned items from the repository* - the items are remote and you need to specify a revision for both.

 **Warning:** The resulting patch file may contain content that was written using a mix of encodings, based upon the encodings of the files that were compared. If you open the generated patch file in a text editor, it may result in unrecognizable content.

#### Generating a Patch - Local Items

Based on a versioned item (already committed or scheduled for addition) in the working copy, you can generate patches that contain the local changes, the differences between a specific revision of that item and the item itself, or differences between the pristine item and another item from the repository. There are four options for generating a patch based upon local items.

To open the **Create patch** wizard, use the  **Create patch...** action from the **Tools** menu or from the contextual menu in the **Modified**, **Incoming**, **Outgoing**, or **Conflicts** modes.



**Figure 371: The Create Patch Wizard - Local Items**

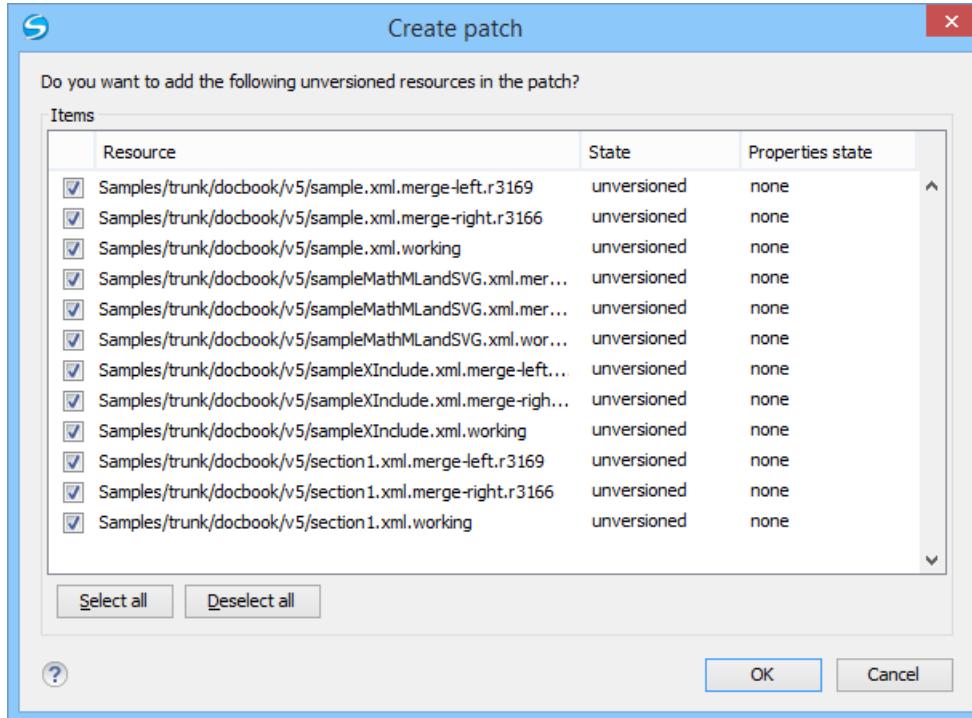
#### Create Patch from Local Modifications

This is the most commonly used type of patch and contains only the local changes for the selected item.

This option is useful if you want to share changes with other team members and you are not yet ready to commit them. This option is only available for local items that contain modifications. It is not available for items in which the local file status is *unversioned or ignored, and in some cases missing or obstructed*.

The steps are as follows:

1. Go to menu **Tools > Create patch**.  
This opens the **Create patch** wizard.
2. Select the **Create patch from local modifications** option in the dialog box.
3. Optionally, if you want *to configure the options* for your patch, press the **Next** button.  
This options page does not remember your selections when creating future patches. It will revert to the default values.  
The **Options** wizard page is displayed.
4. Press the **Create patch** button.  
If the patch is applied on a folder of the working copy and that folder contains *unversioned files*, this step of the wizard offers the option of selecting the ones that will be included in the patch.



**Figure 372: The Create Patch Dialog Box - Add Unversioned Resources**

The patch is created and stored in the path specified in [the Output section of the Options page](#) or in the default location.

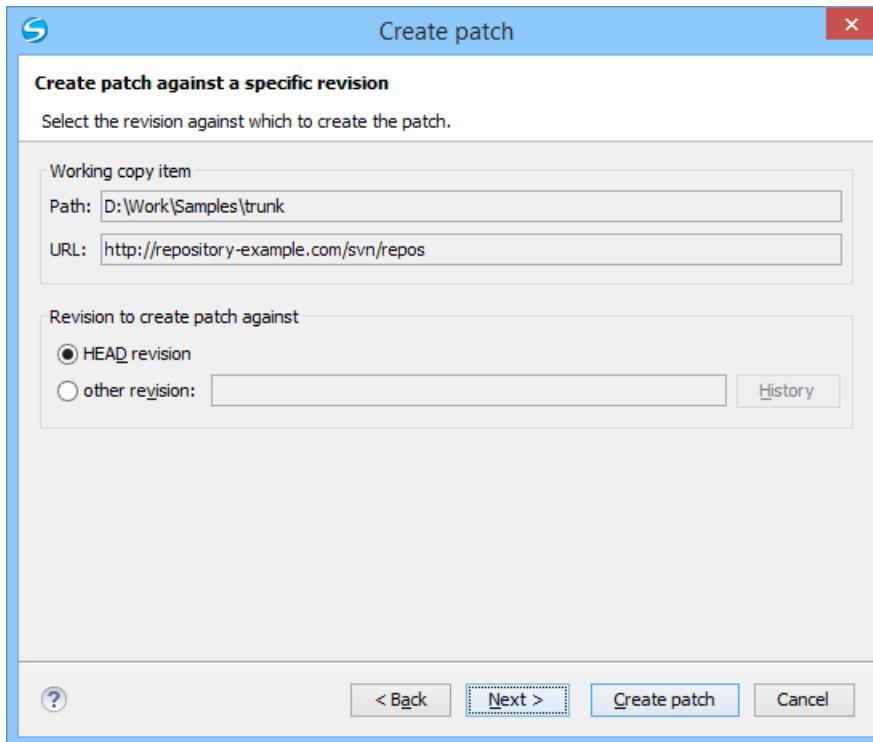
#### Create Patch Against a Specific Revision

This type of patch contains changes between an old revision and the current content from the selected item within the working copy.

This option is useful if you want to obtain differences between an older revision and the current state of the working copy (for instance, to test how current changes apply to an older version).

The steps are as follows:

1. Go to menu **Tools > Create patch**.  
This opens the **Create patch** wizard.
2. Select the **Create patch against a specific revision** option in the dialog box.
3. Press the **Next** button.  
The second step of the wizard is opened:



**Figure 373: The Create Patch Wizard - Step 2**

4. Select the **revision to create patch against**.

You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press [the History button](#) to display a list of the item revisions.

 **Note:** If the **revision to create patch against** is older than the revision to which the working copy item was updated, the patch will include changes that were made **after** the selected revision.

5. Optionally, if you want [to configure the options](#) for your patch, press the **Next** button.

This options page does not remember your selections when creating future patches. It will revert to the default values.

The **Options** wizard page is displayed.

6. Press the **Create patch** button.

The patch is created and stored in the path specified in [the Output section of the Options page](#) or in the default location.

#### Create Patch Between Two Revisions of an Item

This type of patch contains historical changes between two revisions of a selected item.

This option is useful if you want to share changes between two revisions with other team members.

 **Tip:** If you need to generate a patch between two revisions of a previously *deleted*, *moved*, or *replaced* item, you should use [the Create patch between two repository items option](#) instead.

The steps are as follows:

1. Go to menu **Tools > Create patch**.

This opens the **Create patch** wizard.

2. Select the **Create patch between two revisions of an item** option in the dialog box.

3. Press the **Next** button.

The second step of the wizard is opened:



**Figure 374: The Create Patch Wizard - Step 2**

**4.** Select the starting and ending revisions in the **From** and **To** sections.

You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the History button* to display a list of the item revisions.

 **Note:** The patch will only include changes between the two specified revisions, starting with the changes that were made **after** the older revision.

 **Tip:** If you want to reverse changes done between two revisions by using a patch file, you can specify the newer revision in the **From** section and the older version in the **To** section.

**5.** Optionally, if you want *to configure the options* for your patch, press the **Next** button.

This options page does not remember your selections when creating future patches. It will revert to the default values.

The **Options** wizard page is displayed.

**6.** Press the **Create patch** button.

The patch is created and stored in the path specified in *the Output section of the Options page* or in the default location.

### Create Patch Between Two Repository Items

This type of patch contains changes between one version of an item and a specific version of another item.

This option is useful for generating a patch that contains changes between existing, or even previously deleted, moved, or replaced items from different branches. This is the default option when you do not have a working copy loaded, when no repository items are selected, or when exactly two repository items of the same kind are selected.

 **Tip:** To access an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, URL@rev1234).

The steps are as follows:

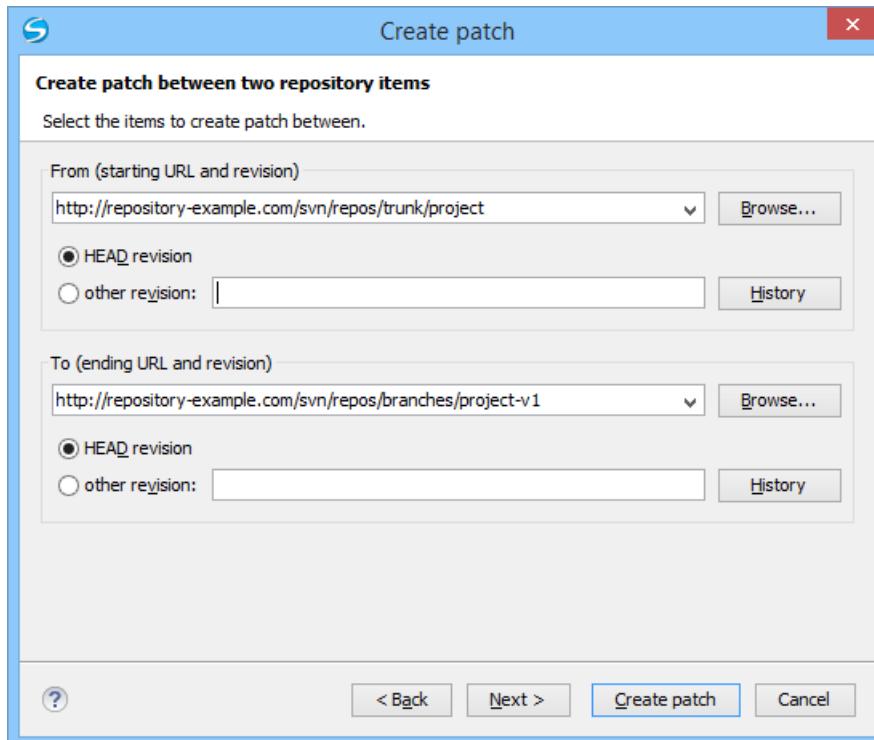
**1.** Go to menu **Tools > Create patch**.

This opens the **Create patch** wizard.

**2.** Select the **Create patch between two repository items** option in the dialog box.

**3.** Press the **Next** button.

The second step of the wizard is opened:



**Figure 375: The Create Patch Wizard - Step 2**

4. Select *the starting and ending URLs* and revisions in the **From** and **To** sections.

You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the History button* to display a list of the item revisions.

 **Important:** Both URLs must point to items from the same repository.

 **Note:** If you use a *peg* revision in the URL field, anything specified in the **other revision** field is ignored.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

This options page does not remember your selections when creating future patches. It will revert to the default values. The **Options** wizard page is displayed.

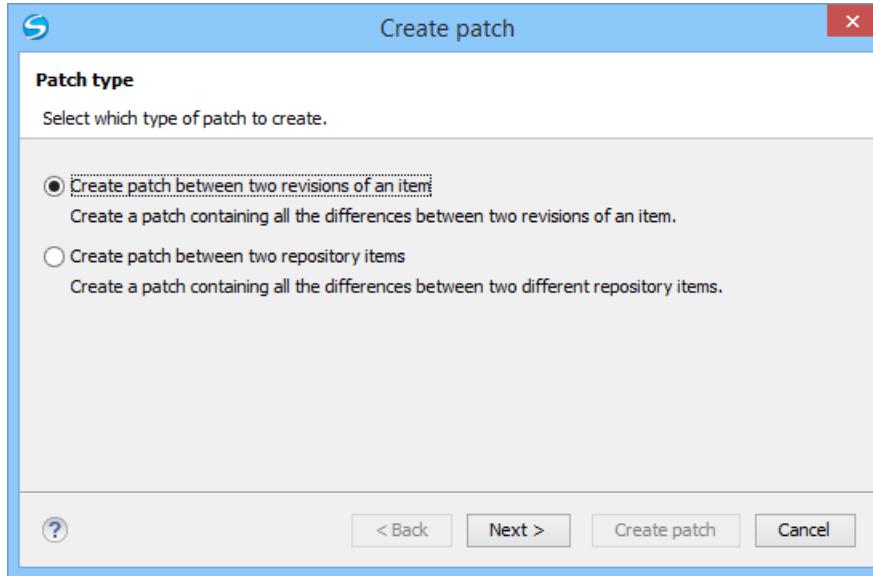
6. Press the **Create patch** button.

The patch is created and stored in the path specified in *the Output section of the Options page* or in the default location.

#### *Generating a Patch - Remote Items*

Based on a repository item, you can generate patches that contain the differences between two specific revisions of that item, or between a revision of that same item and another revision of another item from the repository. There are two options for generating a patch based upon remote items.

To open the **Create patch** wizard, use the  **Create patch...** action from the **Tools** menu.



**Figure 376: The Create Patch Wizard - Remote Items**

#### Create Patch Between Two Revisions of an Item

This type of patch contains historical changes between two revisions of a selected item.

This option is useful if you want to share changes between two revisions with other team members.

- i Tip:** If you need to generate a patch between two revisions of a previously *deleted*, *moved*, or *replaced* item, you should use [the Create patch between two repository items option](#) instead.

The steps are as follows:

1. Go to menu **Tools > Create patch**.  
This opens the **Create patch** wizard.
2. Select the **Create patch between two revisions of an item** option in the dialog box.
3. Press the **Next** button.

The second step of the wizard is opened:



**Figure 377: The Create Patch Wizard - Step 2**

4. Select the starting and ending revisions in the **From** and **To** sections.

You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press [the History button](#) to display a list of the item revisions.

- i Note:** The patch will only include changes between the two specified revisions, starting with the changes that were made **after** the older revision.

- i Tip:** If you want to reverse changes done between two revisions by using a patch file, you can specify the newer revision in the **From** section and the older version in the **To** section.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

This options page does not remember your selections when creating future patches. It will revert to the default values.

The **Options** wizard page is displayed.

6. Press the **Create patch** button.

The patch is created and stored in the path specified in *the Output section of the Options page* or in the default location.

### Create Patch Between Two Repository Items

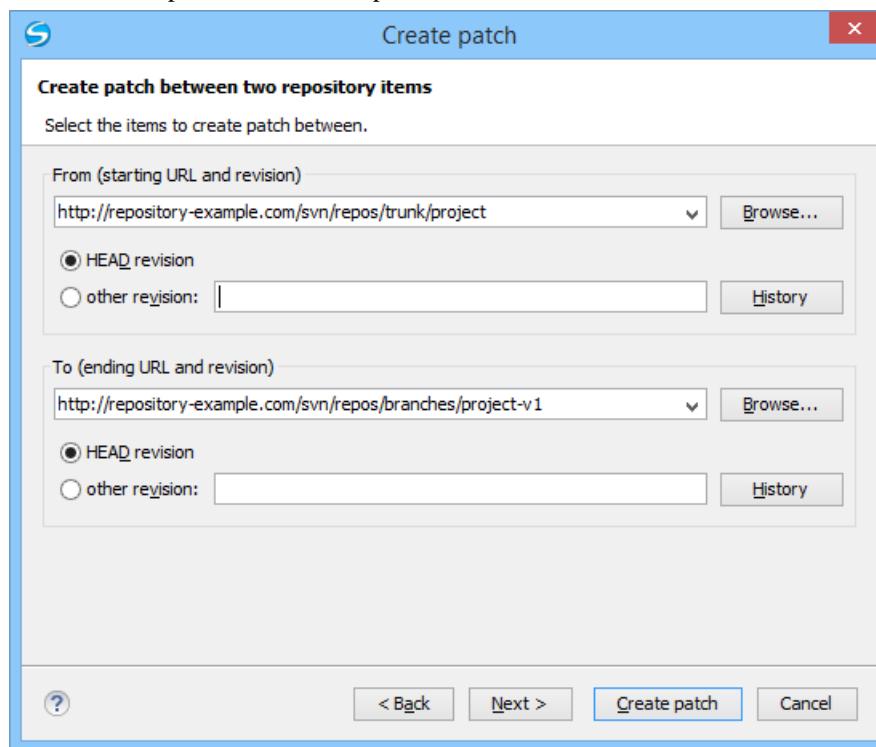
This type of patch contains changes between one version of an item and a specific version of another item.

This option is useful for generating a patch that contains changes between existing, or even previously deleted, moved, or replaced items from different branches. This is the default option when you do not have a working copy loaded, when no repository items are selected, or when exactly two repository items of the same kind are selected.

- Tip:** To access an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, URL@rev1234).

The steps are as follows:

1. Go to menu **Tools > Create patch**.  
This opens the **Create patch** wizard.
2. Select the **Create patch between two repository items** option in the dialog box.
3. Press the **Next** button.  
The second step of the wizard is opened:



**Figure 378: The Create Patch Wizard - Step 2**

4. Select *the starting and ending URLs* and revisions in the **From** and **To** sections.

You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the History button* to display a list of the item revisions.

- Important:** Both URLs must point to items from the same repository.



**Note:** If you use a *peg* revision in the URL field, anything specified in the **other revision** field is ignored.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.  
This options page does not remember your selections when creating future patches. It will revert to the default values.  
The **Options** wizard page is displayed.
6. Press the **Create patch** button.  
The patch is created and stored in the path specified in *the Output section of the Options page* or in the default location.

### Patch Options

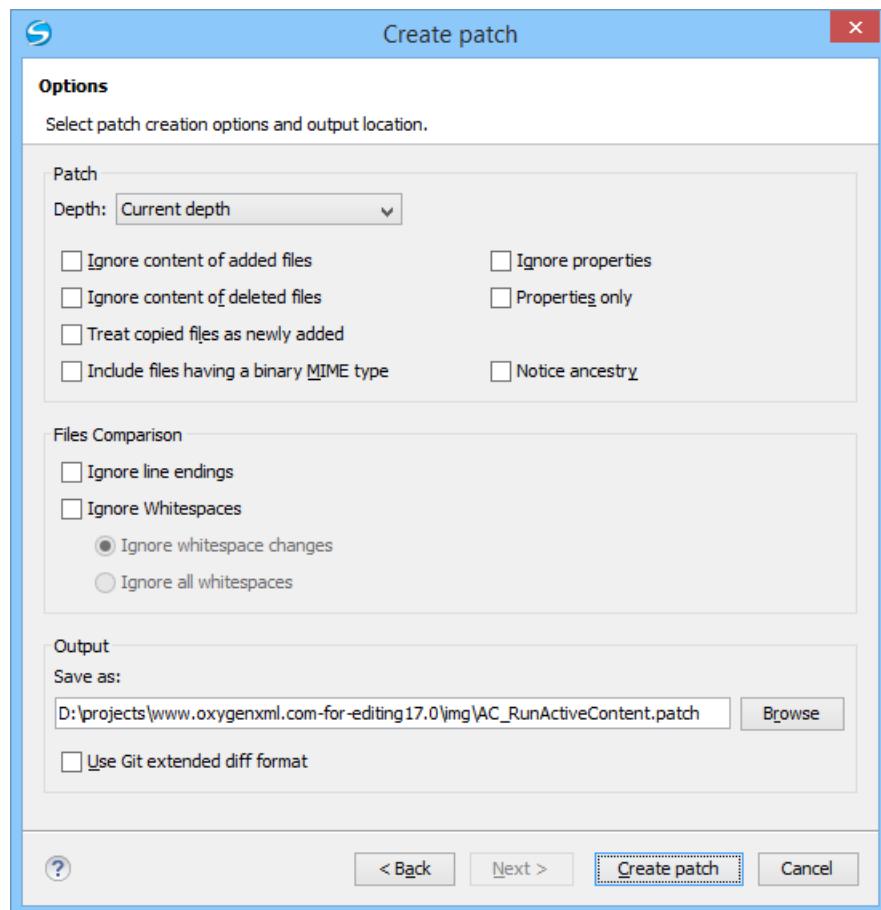


Figure 379: The Create Patch Wizard - Options

### Patch Section

#### Depth - (This option is applicable only for directories)

**Current depth** - The depth of recursing the folder for creating the patch is the same as the depth of that same folder in the working copy (available only when generating patches that contain changes from the working copy).

**Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.

**Immediate children (immediates)** - The patch is created only on the child files and folders without recursing subfolders.

**File children only (files)** - The patch is created only on the child files.

**This folder only (empty)** - The patch is created only on the selected folder (no child file or folder is included in the patch).

### Ignore content of added files

When enabled, the patch file does not include the content of the *added* items. This option corresponds to the `--no-diff-added` option of the `svn diff` command.

### Ignore content of delete files

When enabled, the patch file does not include the content of the *deleted* items. This option corresponds to the `--no-diff-deleted` option of the `svn diff` command.

### Treat copied files as newly added

When enabled, copied items are treated as new, rather than comparing the items with their sources. This option corresponds to the `--show-copies-as-adds` option of the `svn diff` command.

### Include files having a binary MIME type

When enabled, the application is forced to compare items that are considered binary file types. This option corresponds to the `--force` option of the `svn diff` command.

### Ignore properties

When enabled, differences in the properties of items are ignored. This option corresponds to the `--ignore-properties` option of the `svn diff` command.

### Properties only

When enabled, only differences in the properties of the items are included in the patch file (file content is ignored). This option corresponds to the `--properties-only` option of the `svn diff` command.



**Note:** The **Ignore properties** and **Properties only** options are mutually exclusive.

### Notice ancestry

If enabled, the SVN common ancestry is taken into consideration when calculating the differences. This option corresponds to the `--notice-ancestry` option of the `svn diff` command.

## Files Comparison Section

### Ignore line endings

If enabled, the differences in line endings are ignored when the patch is generated. This option corresponds to the `--ignore-eol-style` option of the `svn diff` command.

### Ignore whitespaces

If enabled, it allows you to specify how the whitespace changes should be handled. When enabled, you can then choose between two options:

- **Ignore whitespace changes** (`--ignore-space-change`) - Ignores changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces).



**Note:** Whitespaces that were added where there were none before, or that were removed, are still considered to be changes.

- **Ignore all whitespaces** (`--ignore-all-space`) - Ignores all types of whitespace changes.

## Output Section

### Save as

The patch will be created and saved in the specified file.

### Use Git extended diff format

When enabled, the patch is generated using the *Git* format. This option corresponds to the `--git` option of the `svn diff` command.

## Working with Repositories

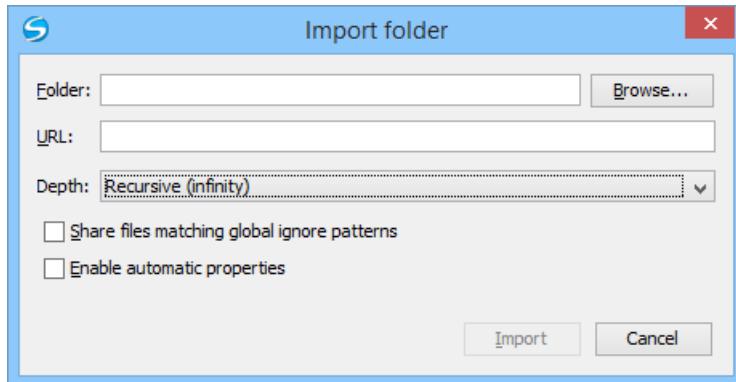
This section explains how to locate and browse SVN repositories in Syncro SVN Client.

## Importing Resources Into a Repository

Importing resources into a repository is the process of copying local files and directories into a repository so that they can be managed by an Apache Subversion™ server. If you have already been using Subversion and you have an existing working copy you want to use, then you will likely want to follow the procedure for [using an existing working copy](#).

The **Import folder...** and **Import Files...** actions are available from the **Import** submenu of the **Repository** menu or of the contextual menu in the **Repositories** view. These actions open a dialog box that allow you to select the directories or files that will be imported into the selected repository location.

The **Import folder...** action opens the **Import folder** dialog box.



**Figure 380: The Import Folder Dialog Box**

You can configure the following options:

### Folder

Specify [the local folder](#) by using the text box or the **Browse** button. This folder should not be empty or already under version control.

**Important:** By default, the SVN system only imports the content of the specified folder, and not the folder itself. Therefore, it is recommended to use the **Browse** button to select the local folder so that the client will automatically append the name of it to the specified URL.

### URL

Specify [the repository location](#) that will be used to access the folder to be imported.

**Note:** *Peg revisions* have no effect for this operation since it is used to send information to the repository.

**Attention:** If the new location already exists, make sure that it is an empty directory to avoid mixing your project content with other files (if items exist with the same name, an error will occur and the operation will not proceed). Otherwise, if the address does not exist, it is created automatically.

### Depth

**Recursive (infinity)** - Imports all the files and folders contained in the selected folder.

**Immediate children (immediates)** - Imports only the child files and folders without recursing subfolders.

**File children only (files)** - Import only the child files.

**This folder only (empty)** - Imports only the selected folder (no child file or folder is included).

### Share files matching global ignore patterns

When enabled, the file names that match the patterns defined in either of the following locations are also imported into the repository:

- The *global-ignores* property in [the SVN configuration file](#).
- The **File name ignore patterns** option in the **SVN > Working Copy preferences page**.

## Enable automatic properties/Disable automatic properties

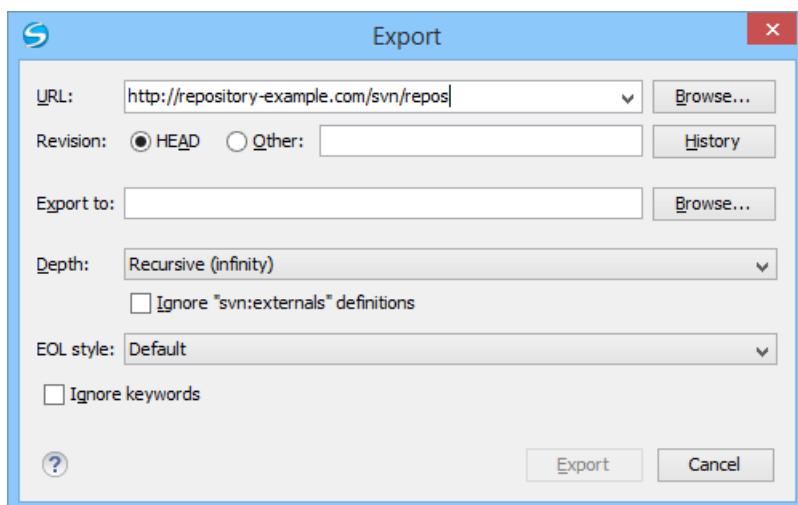
Enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in [the SVN configuration file](#).

 **Note:** This option is available only when there are defined properties to be applied automatically for newly added items under version control. You can define these properties in the SVN config file (in the `auto-props` section). Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable automatic properties**, or **Disable automatic properties**.

## Exporting Resources From a Repository

This is the process of taking a resource from the repository and saving it locally in a clean form, with no version control information. This is very useful when you need a clean build for an installation kit.

The **Export** dialog box is very similar to the **Check out** dialog box:



**Figure 381: Export from Repository Dialog Box**

You can configure the following options:

### URL

Specify [the source directory from the repository](#) by using the text box or the **Browse** button.

 **Tip:** To export an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a [peg revision](#) at the end (for example, URL@rev1234).

 **Note:** The content of the selected directory from the repository and not the directory itself will be exported to the file system.

### Revision

You can choose between the **HEAD** or **Other** revision. If you need to export a specific revision, specify it in the **Other** text box or use the **History** button and choose a revision from the **History** dialog box.

### Export to

Specify [the location where you want to export](#) the repository directory by typing the local path in the text box or by using the **Browse** button. If the specified local path does not point to an existing directory, it will automatically be created.

 **Important:** By default, the SVN system only exports the content of the directory specified by the URL, and not the directory itself. Therefore, it is recommended to use the **Browse** button to select the *export* location so that the client will automatically append the name of the remote directory to the path of the selected directory.



**Warning:** The destination directory should be empty. If files exist, they will be overwritten by exported files with matching names.

## Depth

**Recursive (infinity)** - Exports all the files and folders contained in the selected folder.

**Immediate children (immediates)** - Exports only the child files and folders without recursing subfolders.

**File children only (files)** - Export only the child files.

**This folder only (empty)** - Exports only the selected folder (no child file or folder is included).

## Ignore "svn:externals" definitions

When enabled, external items are ignored in the export operation. This option is only available if you choose the **Recursive (infinity)** depth.

## EOL style

Defines the *end-of-line (EOL)* marker that should be used when exporting files that have the value or the `svn:eol-style` property set to `native`. You can choose between the following styles:

- **Default** - It uses the system-specific *end-of-line* marker.
- **CRLF** - The **Windows**-specific *end-of-line* marker (*carriage return - line feed*).
- **LF** - The **Unix / OS X**-specific *end-of-line* marker (*line feed*).
- **CR** - The **Mac OS 9 (or older)**-specific *end-of-line* marker (*carriage return*).

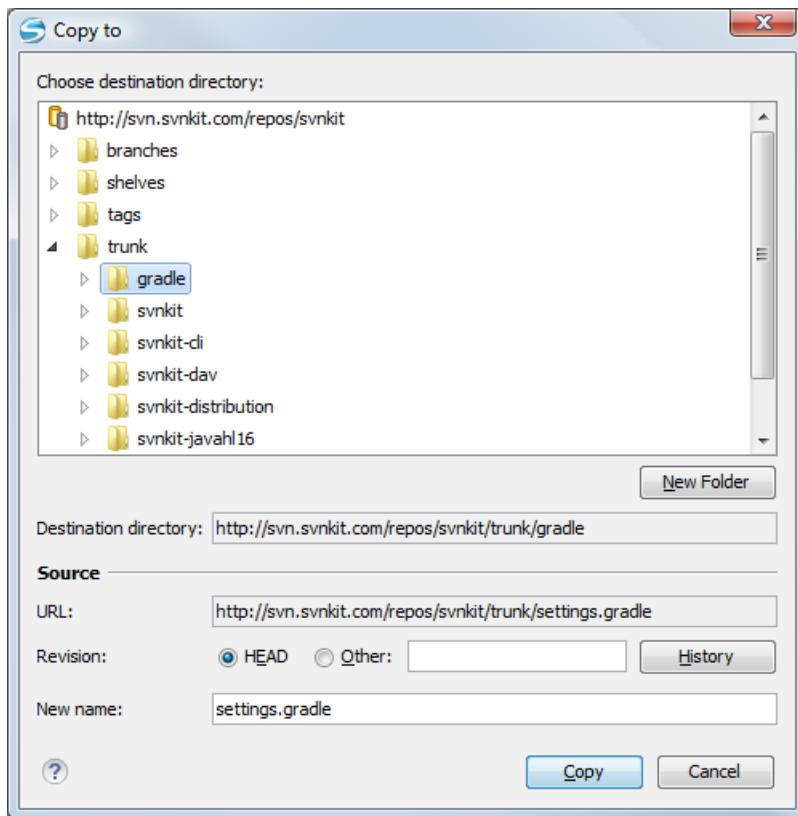
## Ignore keywords

When enabled, the export operation does not expand the *SVN keywords* found inside the files.

## Copy / Move / Delete Resources From a Repository

Once you have a location defined in the [Repositories view](#), you can execute commands like copy, move and delete directly on the repository. The commands correspond to the following actions in the contextual menu:

The **Copy to** and **Move to** action allows you to copy and move individual or multiple resources to a specific directory from the *HEAD* revision of the repository.



**Figure 382: Copy/Move Items in Repository**

The dialog box used to copy or move items allows you to browse the *HEAD* revision of the repository and select the destination of the items, presenting its repository URL below the tree view.

The **Source** section presents relevant options regarding the item(s) that you move or copy:

- **URL** - This field is displayed only if you copy/move a single item.
- **Revision** - Presents the revision from which you copy one or more items, allowing you to also choose another revision.

**Note:** Since only items from the HEAD revision can be moved, the **Revision** options are not presented for the **Move to** action.

**Note:** When you copy a single item while browsing a revision other than *HEAD*, the **Revision** options present this revision but does not allow you to change it. The same applies if copying multiple items.

- **New name** - This option is presented when you copy or move a single item, allowing you to also rename it.

Another useful action is **Delete**, allowing you to erase resources directly from the repository.

All three actions are commit operations and you will be prompted with the **Commit message** dialog box.

### Sparse Checkout

Sometimes you need to check out only certain parts of a directory tree. For this you can check out the top directory (*the action Check out from the Repositories view*) and then update recursively only the needed directories (*the action Update from the Working Copy view*). Now, each directory has a depth set to it, which has four possible values:

- **Recursive (infinity)** - Updates all descendant directories and files recursively.
- **Immediate children (immediates)** - Updates the directory, including direct child directories and files, but does not populate the child directories.
- **File children only (files)** - Updates the directory, including only child files without the child directories.

- **This folder only (empty)** - Updates only the selected directory, without updating any children.

For some operations, you can use as depth the current depth registered on the directories from the working copy (the value **Current depth**). This is the depth value defined in a previous check out or update operation.

The sparse checked out directories are presented in the **Working Copy view** with a marker corresponding to each depth value, in the top left corner, as follows:

-  **Recursive (infinity)** - This is the default value and it has no mark. The directory has no limiting depth.
-  **Immediate children (immediates)** - The directory is limited to direct child directories (without contents) and files.
-  **File children only (files)** - The directory is limited to direct child files only.
-  **This folder only (empty)** - The directory has *empty* depth set.

A depth set on a directory means that some operations process only items within the specified depth range. For example, **Synchronize** on a working copy directory reports the repository modified items within the depth set on the directory and those existing in the working copy outside of this depth.

The depth information is also presented in the **SVN Information** dialog box and in the tool tip displayed when hovering a directory in the **Working Copy** view.

## Syncro SVN Client Views

The main working area occupies the center of the application window, which contains the most important views:

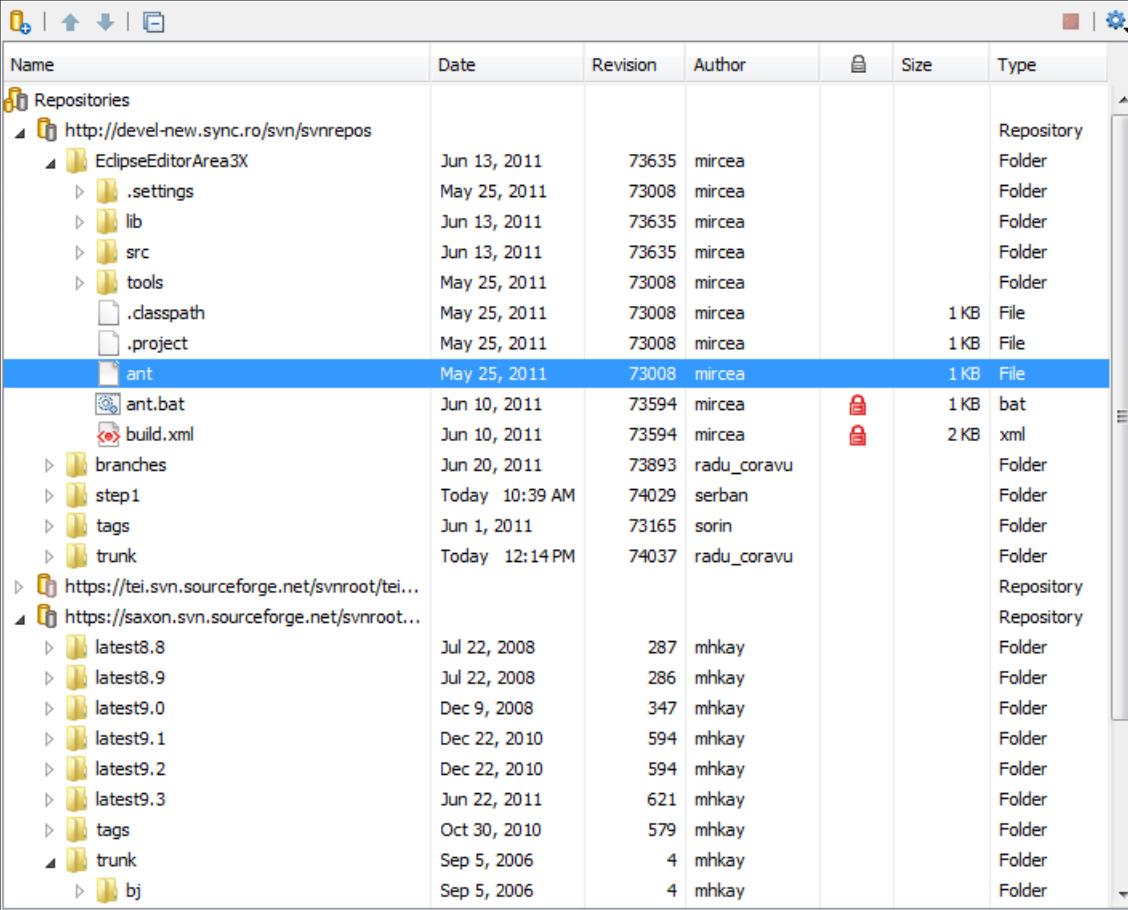
- [Repositories View](#)
- [Working Copy View](#)
- [History View](#)
- [Console View](#)

The other views that support the main working area are also presented in this section.

### Repositories View

The **Repositories** view allows you to define and manage Apache Subversion™ repository locations and browse repositories. If no connections to your repository are available, you can [add a new repository location](#). Repository files and folders are presented in a tree view with the repository locations at the first level, where each location represents a connection to a specific repository. More information about each resource is displayed in a tabular form:

- **Date** - Date when the resource was last modified.
- **Revision** - The revision number at which the resource was last time modified.
- **Author** - Name of the person who made the last modification on the resource.
- **Size** - Resource size on disk.
-  **Lock information** - Information about the lock status of a file. When a repository file is locked by a user the  icon is displayed in this column. If no icon is displayed the file is not locked. The tooltip of this column displays the details about lock:
  - owner - the name of the user who created the lock.
  - date - the date when the user locked the file.
  - expires on - date when the lock expires. Lock expiry policy is set in the repository options, on the server side.
  - comment - the message attached when the file was locked.
- **Type** - Contains the resource type or file extension.



The screenshot shows the 'Repositories' view in the Oxygen XML Editor. The left pane displays a tree view of SVN repositories, while the right pane shows a detailed table of files and folders within a selected repository. The table has columns for Name, Date, Revision, Author, Size, and Type.

Name	Date	Revision	Author		Size	Type
Repositories						
http://devel-new.sync.ro/svn/svnrepos						Repository
EclipseEditorArea3X	Jun 13, 2011	73635	mircea			Folder
.settings	May 25, 2011	73008	mircea			Folder
lib	Jun 13, 2011	73635	mircea			Folder
src	Jun 13, 2011	73635	mircea			Folder
tools	May 25, 2011	73008	mircea			Folder
.classpath	May 25, 2011	73008	mircea		1 KB	File
.project	May 25, 2011	73008	mircea		1 KB	File
ant	May 25, 2011	73008	mircea		1 KB	File
ant.bat	Jun 10, 2011	73594	mircea	🔒	1 KB	bat
build.xml	Jun 10, 2011	73594	mircea	🔒	2 KB	xml
branches	Jun 20, 2011	73893	radu_coravu			Folder
step1	Today 10:39 AM	74029	serban			Folder
tags	Jun 1, 2011	73165	sorin			Folder
trunk	Today 12:14 PM	74037	radu_coravu			Folder
https://tei.svn.sourceforge.net/svnroot/tei...						Repository
https://saxon.svn.sourceforge.net/svnroot...						Repository
latest8.8	Jul 22, 2008	287	mhkay			Folder
latest8.9	Jul 22, 2008	286	mhkay			Folder
latest9.0	Dec 9, 2008	347	mhkay			Folder
latest9.1	Dec 22, 2010	594	mhkay			Folder
latest9.2	Dec 22, 2010	594	mhkay			Folder
latest9.3	Jun 22, 2011	621	mhkay			Folder
tags	Oct 30, 2010	579	mhkay			Folder
trunk	Sep 5, 2006	4	mhkay			Folder
bj	Sep 5, 2006	4	mhkay			Folder

**Figure 383: Repositories View**

## Toolbar

The **Repositories** view's toolbar contains the following buttons:

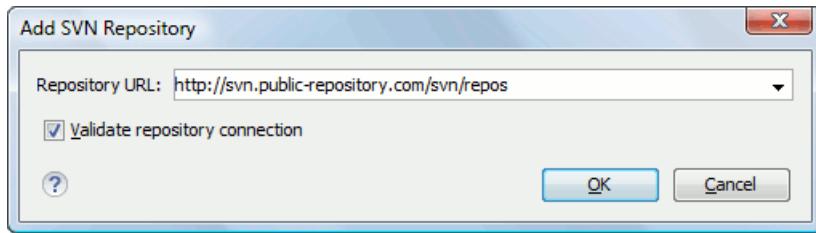
- ✚ **New Repository Location** - Allows you to enter a new repository location by means of the **Add SVN Repository** dialog.
- ⬆ **Move Up** - Move the selected repository up one position in the list of repositories in the **Repositories** view.
- ⬇ **Move Down** - Move the selected repository down one position in the list of repositories in the **Repositories** view.
- [-] **Collapse all** - Collapses all repository trees.
- 🛑 **Stop** - Stops the current repository browsing operation executed when a repository node is expanded. This is useful when the operation takes too long or the server is not responding.
- ⚙ **Settings** - Allows you to configure the resource table appearance.

## Contextual Menu Actions

The **Repositories** view contextual menu contains different actions depending on the selected item. If a repository location is selected, the following management actions are available:

### ✚ New Repository Location... (**Ctrl Alt N (Command Alt N on OS X)**)

Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.



**Figure 384: Add SVN Repository Dialog Box**

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

#### ✳️ **Edit Repository Location... (Ctrl Alt E (Command Alt E on OS X))**

Context-dependent action that allows you to edit the selected repository location using the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.

#### **Change the Revision to Browse... (Ctrl Alt Shift B (Command Alt Shift B on OS X))**

Context-dependent action that allows you to change the selected repository revision using the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.

#### ✖️ **Remove Repository Location... (Ctrl Alt Shift R (Command Alt Shift R on OS X))**

Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.

The following actions are common to all repository resources:

#### **Open**

Opens the selected file in the Editor view in read-only mode.

#### **Open with...**

Displays the **Open with...** dialog to specify the editor in which the selected file is opened. In case multiple files are selected, only external applications can be used to open the files.

#### **Save as...**

Saves the selected files locally, as they are in the browsed revision.

#### ⌚ **Refresh (F5)**

Refreshes the resource selected in the **Repositories** view.

#### ✳️ **Check out... (Ctrl Alt Shift C (Command Alt Shift C on OS X))**

Allows you to create a working copy from a repository directory, on your local file system. To read more about this operation, see the section [Check out a working copy](#).

#### **Branch/Tag...**

Allows you to create a branch or a tag from the selected folder in the repository. To read more about how to create a branch/tag, see the [Creation and management of Branches/Tags](#) section.

#### **Share project...**

Allows you to [share a new project](#) using an SVN repository. The local project is automatically converted into an SVN working copy.

#### **Import:**

##### **Import folder... (Ctrl Alt Shift M (Command Alt Shift M on OS X))**

Allows you to import the contents of a specified folder from the file system into the selected folder in a repository. To read more about this operation, see the section [Importing resources into a repository](#).



**Note:** The difference between the **Import folder...** and **Share project...** actions is that the latter also converts the selected directory into a working copy.

**Import Files... (Ctrl Alt I (Command Alt I on OS X))**

Imports the files selected from the files system into the selected folder in the repository.

**Export...**

Opens [the Export dialog box](#) that allows you to configure options for exporting a folder from the repository to the local file system.

**Show History... (Ctrl H (Command H on OS X))**

Displays the history of the selected resource. At the start of the operation, you can set filtering options.

**Show Annotation... (Ctrl Shift A (Command Shift A on OS X))**

Opens the **Show Annotation** dialog box that computes [the annotations for a file and displays them in the Annotations view](#), along with the history of the file in the **History** view.

**Revision Graph (Ctrl Shift G (Command Shift G on OS X))**

This action allows you to see the graphical representation of a resource history. For more details about a resource revision graph see the section [Revision Graph](#). This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

**Copy URL Location (Ctrl Alt U (Command Alt U on OS X))**

Copies to clipboard the URL location of the selected resource.



Copies to a specified location the currently selected resource(s). This action is also available when you browse other revisions than the latest one (*HEAD*), to allow restoring previous versions of an item.

**Move to... (Ctrl M (Command M on OS X))**

Moves to a specified location the currently selected resource(s).

**Rename... ((F2))**

Renames the selected resource.

**✖ Delete ((Delete))**

Deletes selected items from the repository via an immediate commit.

**New Folder... (Ctrl Shift F (Command Shift F on OS X))**

Allows you to create a folder in the selected repository path (available only for folders).

**Locking**

(available only for files):

**Lock... (Ctrl K (Command K on OS X))**

Allows you to lock certain files for which you need exclusive access. For more details on the use of this action, see [Locking a file](#).

**Unlock... (Ctrl Shift K (Command Shift K on OS X))**

Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

**Show SVN Properties (Ctrl Shift P (Command Shift P on OS X))**

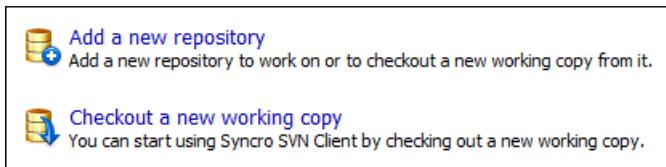
Brings up the [Properties view](#) displaying the SVN properties for the selected resource. This view does not allow adding, editing, or removing SVN properties of a repository resource. These operations are allowed only for working copy resources.

**Show SVN Information (Ctrl I (Command I on OS X))**

Provides additional information for the selected resource. For more details, go to [Obtain information for a resource](#).

**Assistant Actions**

When there is no repository configured, the **Repositories** view mode lists the following two actions:



## Drag and Drop Operations

The structure of the files tree can be changed with drag and drop operations inside the **Repositories** view. These operations behave in the same way with the **Copy to/Move to** operations.

## Working Copy View

The **Working Copy** view allows you to manage the content of an SVN working copy.

The toolbar contains:

- the list of defined working copies
- a set of view modes that allow you to filter the content of the working copy based on the resource status (like incoming or outgoing changes)
- Settings** menu

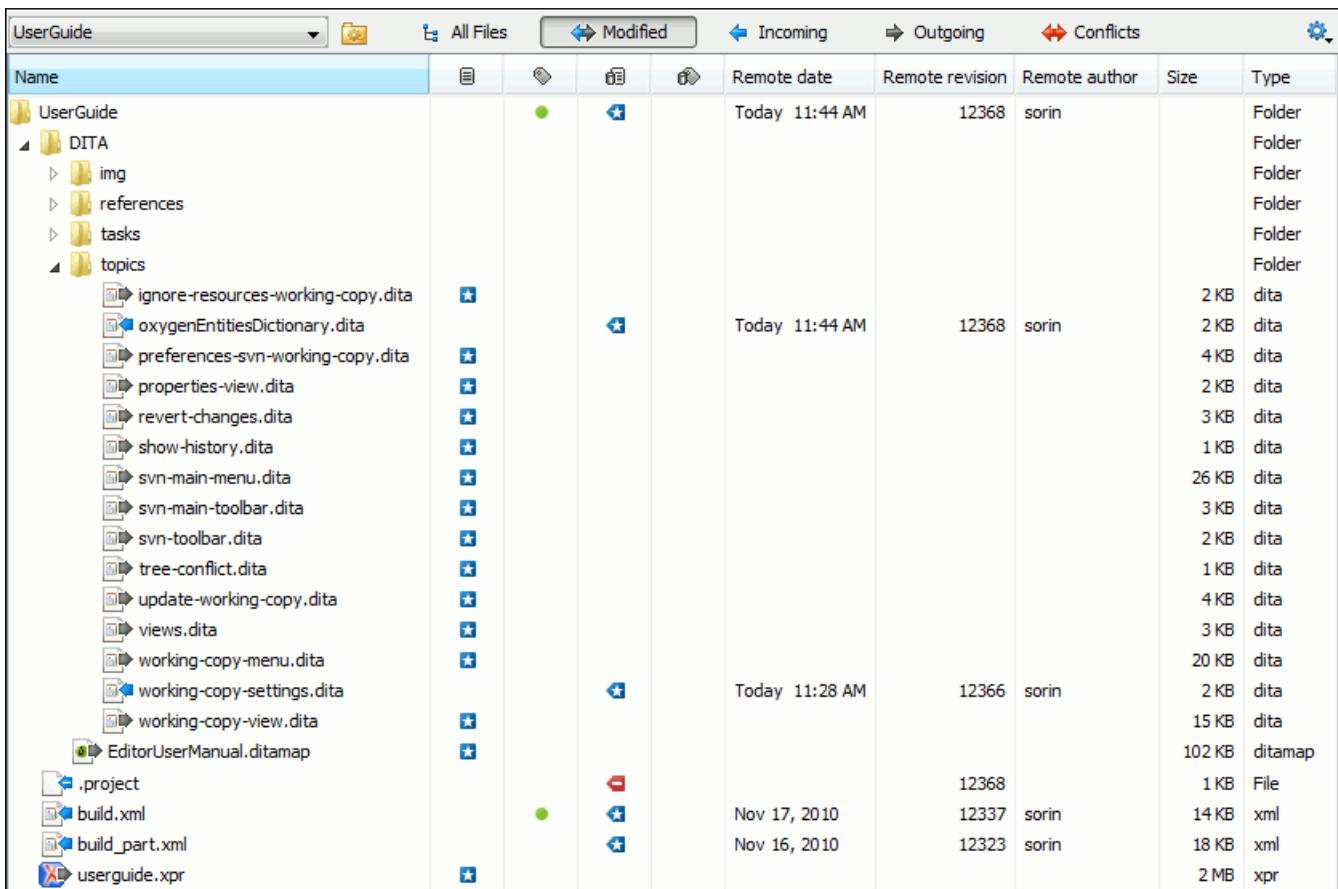
If you click any of the view modes (**All Files**, **Modified**, **Incoming**, **Outgoing**, **Conflicts**), the information displayed changes as follows:

- All Files** - Resources (files and folders) are presented in a hierarchical structure with the root of the tree representing the location of the working copy on the file system. Each resource has an icon representation which describes the type of resource and also depicts the state of that resource with a small overlay icon.

Name	Date	Revision	Author	Size	Type
E:\svnkit					File Folder
gradle					File Folder
wrapper					File Folder
gradle-wrapper.jar	May 4, 2011	7618	alex	12 KB	Executable ...
gradle-wrapper.properties	May 4, 2011	7623	alex	1 KB	PROPERTIE...
svnkit	May 15, 2011	7636	alex		File Folder
svnkit-dl					File Folder
.settings	May 10, 2011	7630	alex		File Folder
src					File Folder
main					File Folder
conf	May 4, 2011	7618	alex		File Folder
java	May 4, 2011	7622	alex		File Folder
resources	May 4, 2011	7618	alex		File Folder
scripts	May 10, 2011	7630	alex		File Folder
jsvn	May 4, 2011	7618	alex	2 KB	File
jsvn.bat	May 10, 2011	7630	alex	2 KB	Windows B...
jsvnsetup.openvms	May 4, 2011	7618	alex	1 KB	OPENVMS File
build.gradle	May 4, 2011	7618	alex	2 KB	GRADLE File
svnkit-dav	May 4, 2011	7620	alex		File Folder
svnkit-distribution	May 4, 2011	7623	alex		File Folder
svnkit-javahl16	May 4, 2011	7618	alex		File Folder
svnkit-osgi	May 4, 2011	7623	alex		File Folder
svnkit-test					File Folder
.settings	May 12, 2011	7635	alex		File Folder
configurations	May 4, 2011	7618	alex		File Folder

Figure 385: Working Copy View - All Files View Mode

- **Modified** - The resource tree presents resources modified locally (including those with conflicting content) and remotely. Decorator icons are used to differentiate between various resource states:
  - - incoming modification from repository:
    - - file content or properties modified remotely
    - - new file added remotely
    - - file deleted remotely
  - - outgoing modification to repository:
    - - file content or properties modified locally
    - - new file added locally
    - - file deleted locally
  - - pseudo-conflict state - a resource being locally and remotely modified at the same time, or a parent directory of such a resource.
  - - real conflict state - a resource that had both incoming and outgoing changes and not all the differences could be merged automatically through the update operation (manually editing the local file is necessary for resolving the conflict).



The screenshot shows the Oxygen XML Editor's Working Copy View in Modified View Mode. The interface includes a toolbar with buttons for All Files, Modified, Incoming, Outgoing, Conflicts, and a gear icon. The main area displays a hierarchical tree of files and folders under 'UserGuide'. Each item in the tree has a small icon indicating its status: green for incoming modifications, blue for outgoing modifications, and red for conflicts. To the right of the tree is a detailed table with columns for Name, Type, Remote date, Remote revision, Remote author, Size, and Type. The table lists various DITA files, a ditamap, and XML files, along with their respective details like size and author.

Name	Type	Remote date	Remote revision	Remote author	Size	Type
UserGuide	Folder					
DITA	Folder					
img	Folder					
references	Folder					
tasks	Folder					
topics	Folder					
ignore-resources-working-copy.dita	dita				2 KB	
oxygenEntitiesDictionary.dita	dita				2 KB	
preferences-svn-working-copy.dita	dita				4 KB	
properties-view.dita	dita				2 KB	
revert-changes.dita	dita				3 KB	
show-history.dita	dita				1 KB	
svn-main-menu.dita	dita				26 KB	
svn-main-toolbar.dita	dita				3 KB	
svn-toolbar.dita	dita				2 KB	
tree-conflict.dita	dita				1 KB	
update-working-copy.dita	dita				4 KB	
views.dita	dita				3 KB	
working-copy-menu.dita	dita				20 KB	
working-copy-settings.dita	dita				2 KB	
working-copy-view.dita	dita				15 KB	
EditorUserManual.ditamap	ditamap				102 KB	
.project	File				1 KB	
build.xml	xml				14 KB	
build_part.xml	xml				18 KB	
userguide.xpr	xpr				2 MB	

Figure 386: Working Copy View - Modified View Mode

- **Incoming** - The resource tree presents only incoming changes.
- **Outgoing** - The resource tree presents only outgoing changes.
- **Conflicts** - The resource tree presents only conflicting changes (real conflicts and pseudo-conflicts).

The following columns provide information about the resources:

- **Name** - Resource name. Resource icons can have the following decorator icons:
  - Additional status information:
    - **Propagated modification marker** - A folder marked with this icon indicates that the folder itself presents some changes (like modified properties) or a child resource has been modified.
    - **External** - This indicates a mapping of a local directory to the URL of a versioned resource. It is declared with a `svn:externals` property in the parent folder and it indicates a working copy not directly related with the parent working copy that defines it.
    - **Switched** - This indicates a resource that has been switched from the initial repository location to a new location within the same repository. The resource goes to this state as a result of *the Switch action* executed from the contextual menu of the Working Copy view.
    - **Grayed** - A resource with a grayed icon but no overlaid icon is an ignored resource. It is obtained with the **Add to svn:ignore** action.
  - Current SVN depth of a folder:
    - **Immediate children (immediates)** (a variant of *sparse checkout*) - The directory contains only direct file and folder children. Child folders ignore their content.
    - **File children only (files)** (a variant of *sparse checkout*) - The directory contains only direct file children, disregarding any child folders.
    - **This folder only (empty)** (a variant of *sparse checkout*) - The directory discards any child resource.



#### Note:

- Any folder not marked with one of the depth icons, has recursive depth (*infinity*) set by default (presents all levels of child resources).
- Although folders not under version control can have no depth set, Oxygen XML Editor presents *unversioned* and *ignored* folders with *empty* depth when **Show unversioned directories content** or **Show ignored directories content** options are disabled.
- **Local file status** - Shows the changes of working copy resources that were not committed to the repository yet. The following icons are used to mark resource status:
  - - Resource is *not under version control (unversioned)*.
  - - Resource is being *ignored* because it is not under version control and its name matches a file name pattern defined in one of the following places:
    - *global-ignores* section in the SVN client-side *config file*.
  - **Attention:** If you don't explicitly set the *global-ignores* runtime configuration option - either to your preferred set of patterns or to an empty string - Subversion uses the default value.
  - *Application global ignores option* of Oxygen XML Editor.
  - the value of a *svn:ignore property* set on the parent folder of the resource being ignored.

- - Marks a newly created resource, *scheduled for addition* to the version control system.
- - Marks a resource *scheduled for addition*, created by copying a resource already under version control and inheriting all its SVN history.
- - The content of the resource has been *modified*.
- - Resource has been *replaced* in your working copy (the file was scheduled for deletion, and then a new file with the same name was scheduled for addition in its place).
- - Resource is *deleted*(scheduled for deletion from **Repository** upon the next commit).
- - The resource is *incomplete* (as a result of an interrupted *check out* or *update* operation).
- - The resource is *missing* because it was moved or deleted without using an SVN-aware application.
- - The contents of the resource is in *real conflict state*.
- - Resource is in a *name conflict* state.
- - Resource is in *tree conflict* state after an update operation because:
  - Resource was locally modified and incoming deleted from repository.
  - Resource was locally scheduled for deletion and incoming modified.
- **Local properties status** - Marks the resources that have SVN properties, with the following possible states:
  - - The resource has SVN properties set.
  - - The resource properties have been modified.
  - - Properties for this resource are in *real conflict* with property updates received from the repository.
- **Revision** - The current revision number of the resource.
- **Date** - Date when the resource was last time modified on the disk.
- **BASE Revision** - The revision number of the pristine version of the resource.
- **BASE Date** - Date when the pristine version of the resource was last time committed in the repository.
- **Author** - Name of the person who made the last modification on the pristine version of the resource.
- **Remote file status** - Shows changes of resources recently modified in the repository. The following icons are used to mark incoming resource status:
  - - Resource is newly added in repository.
  - - The content of the resource has been modified in repository.
  - - Resource was replaced in repository.
  - - Resource was deleted from repository.
- **Remote properties status** - Resources marked with the icon have incoming modified properties from the repository.
- **Remote revision** - Revision number of the resource latest committed modification.
- **Remote date** - Date of the resource latest modification committed on the repository.
- **Remote author** - Name of the author who committed the latest modification on the repository.
- **Lock information** - Shows the lock state of a resource. The lock mechanism is a convention intended to help you signal other users that you are working with a particular set of files. It minimizes the time and effort wasted in solving possible conflicts generated by clashing commits. A lock gives you exclusive rights over a file, only if other users follow this convention and they do not try to bypass the lock state of a file.

A folder can be locked only by the SVN client application, completely transparent to the user, if an operation in progress was interrupted unexpectedly. As a result, folders affected by the operation are marked with the symbol. To clear the locked state of a folder, use the **Clean up** action.



**Note:** Users can lock only files.

The following lock states are displayed:

- *no lock* - the file is not locked. This is the default state of a file in the SVN repository.
- *remotely locked* ( ) - shown when:
  - another user has locked the file in the repository.
  - the file was locked by the same user from another working copy.
  - the file was locked from the **Repositories** view.

If you try to commit a new revision of the file to the repository, the server does not allow you to bypass the file lock.



**Note:** To commit a new revision, you need to wait for the file to be unlocked. Ultimately, you might try to *break* or *steal* the lock, but this is not what other users expect. Use these actions carefully, especially when you are not the file lock owner.

- *locked* ( ) - displayed after you have locked a file from the current working copy. Now you have exclusive rights over the corresponding file, being the only one who can commit changes to the file in the repository.
- **Note:** Working copies keep track of their locked files, so the locks are presented between different sessions of the application. Synchronize your working copy with the repository to make sure that the locks are still valid (not *stolen* or *broken*).
- *stolen* ( ) - a file already locked from your working copy is being locked by another user. Now the owner of the file lock is the user who stole the lock from you.
- *broken* ( ) - a file already locked from your working copy is no longer locked in the repository (it was unlocked by another user).



**Note:** To remove the *stolen* or *broken* states from your working copy files, you have to **Update** them.

If one of your working copy files is locked, hover the mouse pointer over the lock icon to see more information:

- lock type - current file lock state
- owner - the name of the user who created the lock
- date - the date when the user locked the file
- expires on - date when the lock expires. Lock expiry policy is set in the repository options, on the server side
- comment - the message attached when the file was locked
- **Size** - Resource size on disk
- **Type** - Contains the resource type or file extension



**Note:** The working copy table allows you to show or hide any of its columns and also to sort its contents by any of the displayed columns. The table header provides a contextual menu which allows you to customize the displayed information.

The toolbar allows you to switch between two working copies:

- Drop down list - Contains all the working copies Oxygen XML Editor is aware of. When you select another working copy from the list, the newly selected working copy content is scanned and displayed in the **Working Copy** view.
- ( on Mac OS X) **Working Copies Manager** - opens a dialog box that displays the working copies Oxygen XML Editor is aware of. In this dialog box, you can add existing working copies or remove those you no longer need. If you try to add a folder which is not a valid Subversion working copy, Oxygen XML Editor warns you that the selected directory is not under version control.



**Note:** Removing a working copy from this dialog does NOT remove it from your file system; you will have to do that manually.

## Working Copy Settings

The **Settings** button from the toolbar of the **Working Copy** view provides the following options:

- **Show unversioned directories content** - displays the content of unversioned directories;
 

**Note:** In case this option is disabled, it will be ignored for items that, after a synchronize, are reported as incoming from the repository. This applies for all working copy modes, except **All Files**.
- **Show ignored items** - displays the ignored resource when **All Files** mode is selected;
- **Show ignored directories content** - displays the content of ignored directories when **All Files** mode is selected;
 

**Note:** Although *ignored* items are not presented in the **Modified**, **Incoming**, and **Conflicts** modes, they will be if, after a synchronize, they are reported as incoming from the repository.
- **Show deleted items** - displays the deleted resource when **All Files** mode is selected. All other modes always display deleted resources, disregarding this option;
- **Tree / Compressed / Flat** - affect the way information is displayed inside the **Modified**, **Incoming**, **Outgoing**, and **Conflicts** view modes;
- **Configure columns** - allows you to customize the structure of the **Working Copy** view data. This action opens the following dialog box:

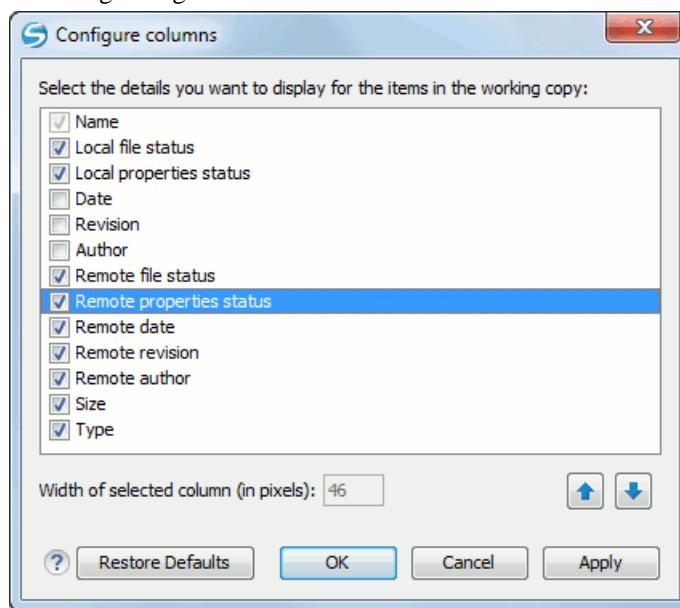


Figure 387: Configure Columns of Working Copy View

The order of the columns can be changed with the two arrow buttons. The column size can be edited in the **Width of selected column** field. The **Restore Defaults** button reverts all columns to the default order, width and enabled/disabled state from the installation of the application.

## Working Copy Format

When an SVN working copy is loaded, Syncro SVN Client first checks the format of the working copy:

- If the format is older than SVN 1.7, you are prompted to upgrade it to SVN 1.8 in order to load it.
- If the format is 1.7, Syncro SVN Client takes into account the state of the **When loading an old format working copy** option.

To change how working copy formats are handled, [open the Preferences dialog box](#), go to **SVN > Working copy**, and configure the options in the [Administrative area](#) section.



#### Note:

- The format of the working copy can be downgraded or upgraded at any time with the **Upgrade** and **Downgrade** actions available in the **Tools** menu. These actions allow switching between SVN 1.7 and SVN 1.8 working copy formats.
- SVN 1.7 working copies cannot be downgraded to older formats.

### Refresh a Working Copy

A refresh is a frequent operation triggered automatically when you switch between two working copies using the toolbar selector of the **Working Copy** view and when you switch between Oxygen XML Editor and other applications.

The **Working Copy** view features a fast refresh mechanism: the content is cached locally when loading the working copy for the first time. Later on, when the same working copy is displayed again, the application uses this cache to detect the changes between the cached content and the current content found on disk. The refresh operation is run on these changes only, thus improving the response time. Improvement is noticeable especially when working with large working copies.

### Contextual Menu Actions

The contextual menu in the **Working Copy** view contains the following actions:

#### **Edit conflict...** ([Ctrl \(Command on OS X\) + E](#))

Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see [Edit conflicts](#).

#### **Open in Compare Editor** ([Ctrl \(Command on OS X\) + Alt + C](#))

Displays changes made in the currently selected file.

#### **Open** ([Ctrl \(Command on OS X\) + O](#))

Opens the selected resource from the working copy. Files are opened with an internal editor or an external application associated with that file type, while folders are opened with the default file system browsing application (e.g. Windows Explorer on Windows, Finder on OS X, etc).

#### **Open with...**

Submenu that allows you to open the selected resource either with Oxygen XML Editor or with another application.

#### **Show in Explorer/Show in Finder**

Opens the parent directory of the selected working copy file and selects the file.



#### **Expand all** ([Ctrl \(Command on OS X\) + Alt + X](#))

Displays all descendants of the selected folder. The same behavior is obtained by double-clicking on a collapsed folder.



#### **Refresh** ([F5](#))

Re-scans the selected resources recursively and refreshes their status in the working copy view.



#### **Synchronize** ([Ctrl \(Command on OS X\) + Shift + S](#))

Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the [Always switch to 'Modified' mode](#) option is selected.

#### **Update** ([Ctrl \(Command on OS X\)+ U](#))

Updates the selected resources to the **HEAD** revision (latest modifications) from the repository. If the selection contains a directory, it will be updated depending on its depth.

#### **Update to revision/depth...**

Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the [sparse checkouts](#) section.

## Commit...

Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what resources to commit. A directory will always be committed recursively. Unversioned resources will be deselected by default. In the **Commit** dialog you can also enter a comment before sending your changes to the repository.

### **Revert...** (**Ctrl (Command on OS X) + Shift + V**)

Undoes all local changes for the selected resources. It does not contact the repository and the files are obtained from Apache Subversion™ pristine copy. It is enabled only for modified resources. See [Revert your changes](#) for more information.

## Override and Update...

Drops any outgoing change and replaces the local resource with the HEAD revision. This action is available on resources with outgoing changes, including conflicting ones. See the [Revert your changes](#) section.

## Override and Commit...

Drops any incoming changes and sends your local version of the resource to the repository. This action is available on conflicting resources. For more information see [Drop incoming modifications](#).

### **Mark Resolved** (**Ctrl (Command on OS X) + Shift + R**)

Instructs the Subversion system that you resolved a conflicting resource. For more information, see [Merge conflicts](#).

### **Mark as Merged** (**Ctrl (Command on OS X) + Shift + M**)

Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the [Merge conflicts](#) section for more information about how you can solve the pseudo-conflicts.

### **Create patch...** (**Ctrl (Command on OS X) + Alt + P**)

Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see [the section about patches](#).

## Compare with:

- **Latest from HEAD** (**Ctrl (Command on OS X) + Alt + H**) - Performs a 3-way diff operation between the selected file and the *HEAD* revision from the repository and displays the result in the **Compare** view. The common ancestor of the 3-way diff operation is the *BASE* version of the file from the local working copy.
- **BASE revision** (**Ctrl (Command on OS X) + Alt + C**) - Compares the working copy file with the *BASE* revision file (the so-called *pristine copy*).
- **Revision** (**Ctrl (Command on OS X) + Alt + R**) - Shows the **History** view containing the log history of that resource.
- **Branch/Tag** - Opens the **Compare with Branch/Tag** dialog box that allows you to specify [another file from the repository](#) (**To URL** field) to compare with the working copy file. You can specify the revision of the repository file by choosing between **HEAD revision** or specific **Other revision**.

 **Tip:** To compare with a file that was deleted, moved, or replaced, you need to specify the original URL (before the file was removed) and use a [peg revision](#) at the end (for example, URL@rev1234).

- **Each other** - Compares two selected files with each other.

These *compare* actions are enabled only if the selected resource is a file.

## Replace with:

- **Latest from HEAD** - Replaces the selected resources with their versions from the *HEAD* revision of the repository.
- **BASE revision** - Replace the selected resources with their versions from the pristine copy (the *BASE* revision).



**Note:** In some cases it is impossible to replace the currently selected resources with their versions from the **BASE/HEAD** revision:

- For the **Replace with BASE revision** action, the resources being unversioned or added have no **BASE** revision, and thus cannot be replaced. However, they will be deleted if the action is invoked on a parent folder. The action will never work for missing folders or for obstructing files (folders being obstructed by a file), since you cannot recover a tree of folders
- For the **Replace with latest from HEAD** action, you must be aware that there are cases when resources will be completely deleted or reverted to the **BASE** revision and then updated to a **HEAD** revision to avoid conflicts. These cases are:
  - the resource is *unversioned, added, obstructed, or modified*
  - the resource is affected by a `svn:ignore` or `svn:externals` property that is locally added on the parent folder and not yet committed to the repository



#### **Show History... (Ctrl (Command on OS X) + H)**

Displays the **History view** where the log history for the selected resource will be presented. For more details about resource history, see the sections about *the resource history view* and *requesting the history for a resource*.



#### **Show Annotation... (Ctrl Shift A (Command Shift A on OS X))**

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the Annotations view*, along with the history of the file in the **History** view.



#### **Revision Graph (Ctrl (Command on OS X) + G)**

This action allows you to see the graphical representation history of a resource. For more details about the revision graph of resources, see *Revision Graph*.

#### **Copy URL Location (Ctrl (Command on OS X) + Alt + U)**

Copies the encoded URL of the selected resource from the Working Copy to the clipboard.

#### **Mark as copied**

You can use this action to mark an item from the working copy as a copy of another item under *version control*, when the copy operation was performed outside of an SVN client. The **Mark as copied** action is available when you select two items (both the new item and source item), and it depends on the state of the source item.

#### **Mark as moved**

You can use this action to mark an item from the working copy as being moved from another location of the working copy, when the move operation was performed outside of an SVN client. The **Mark as moved** action is available when you select two items from different locations (both the new item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

#### **Mark as renamed**

You can use this action to mark an item from the working copy as being renamed outside of an SVN client. The **Mark as renamed** action is available when you select two items from the same directory (both the new item and the source item that is usually reported as *missing*), and it depends on the state of the source item.



#### **Copy to...**

Copies the currently selected resource to a specified location.

#### **Move to... (Ctrl M (Command M on OS X))**

Moves the currently selected resource to a specified location.

#### **Rename... (F2)**

As with the move command, a copy of the original resource will be made with the new name and the original will be marked as deleted. Note that you can only rename one resource at a time.

#### **Delete (Delete)**

Schedules selected items for deletion upon the next commit and removes them from the disk. Depending on the state of each item, you are prompted to confirm the operation.

**New:****New File...**

Creates a new file inside the selected folder. The newly created file will be added under version control only if the parent folder is already versioned.

**New Folder... (Ctrl (Command on OS X)+ Shift + F)**

Creates a child folder inside the selected folder. The newly created folder will be added under version control only if its parent is already versioned.

**New External Folder... (Ctrl (Command on OS X) + Shift + W)**

This operation allows you to add a new external definition on the selected folder. An external definition is a mapping of a local directory to *a URL of a versioned directory*, and ideally a particular revision, stored in the `svn:externals` property of the selected folder.



**Tip:** You can specify a particular revision of the external item by using a *peg revision* at the end of the URL (for example, `URL@rev1234`). You can also use peg revisions to access external items that were deleted, moved, or replaced.

The URL used in the external definition format can be relative. You can specify the repository URL that the external folder points to by using one of the following relative formats:

- `./` - Relative to the URL of the directory on which the `svn:externals` property is set.
- `^/` - Relative to the root of the repository in which the `svn:externals` property is versioned.
- `//` - Relative to the scheme of the URL of the directory on which the `svn:externals` property is set.
- `/` - Relative to the root URL of the server in which the `svn:externals` property is versioned.



**Important:** To change the target URL of an external definition, or to delete an external item, do the following:

1. Modify or delete the item definition found in the `svn:externals` property that is set on the parent folder.
2. For the change to take effect, use the **Update** operation on the parent folder of the external item.



**Note:** Syncro SVN Client does not support definitions of local relative external items.

**Add to "svn:ignore" ... (Ctrl (Command on OS X) + Alt + I)**

Allows you to add files that should not participate in the *version control* operations inside your working copy . This action can only be performed on resources not under *version control*. It actually modifies the value of the `svn:ignore` property in the parent directory of the resource. Read more about this in the [Ignore Resources Not Under Version Control](#) section.

**Add to version control... (Ctrl (Command on OS X) + Alt + V)**

Allows you to add resources that are not under *version control*. For further details, see [Add Resources to Version Control](#) section.

**Remove from version control**

Schedules selected items for deletion from repository upon the next commit. The items are not removed from the file system after committing.

**Clean up (Ctrl (Command on OS X) + Shift + C)**

Performs a maintenance cleanup operation on the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. This is useful when you already know where the problem originated and want to fix it as quickly as possible. It is only active for resources under *version control*.

**Locking:**

- **Scan for locks... (Ctrl (Command on OS X) + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. This is only active for resources under *version control*. For more details see [Scanning for locks](#).

-  **Lock...** (**Ctrl (Command on OS X) + K**) - Allows you to lock certain files that need exclusive access. You can write a comment describing the reason for the lock and you can also force (*steal*) the lock. This action is active only on files under *version control*. For more details on the use of this action see [Locking a file](#).
-  **Unlock...** (**Ctrl (Command on OS X) + Alt + K**) - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

### **Show SVN Properties** (**Ctrl P (Command P on OS X)**)

Brings up the [Properties view](#) and displays the SVN properties for the selected resource.

### **Show SVN Information** (**Ctrl I (Command I on OS X)**)

Provides additional information for the selected resource from the working copy. For more details, go to [Obtain information for a resource](#).

## Drag and Drop Operations

The structure of the files tree can be changed with drag and drop operations inside the **Working Copy** view. These operations behave in the same way with the **Copy to/Move to** operations.

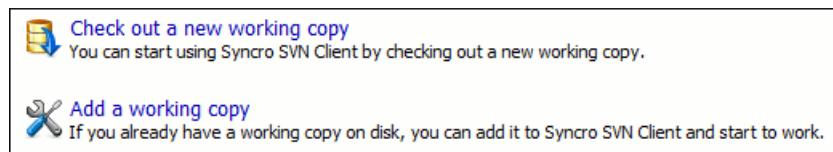
Also, files and folders can be added to the file tree of the view as *unversioned* resources by drag and drop operations from other applications (for example from Windows Explorer or Mac OS X Finder). In this case, the items from the file system are only copied, without removing them from their original location.

 **Attention:** When you drag items from the working copy to a different application, the performed operation is controlled by that application. This means that the moved items are left as *missing* in the working copy (items are moved in the file system only, but no SVN versioning meta-data is changed).

## Assistant Actions

To ensure a continuous and productive work flow, when a view mode has no files to present, it offers a set of guiding actions with some possible paths to follow.

Initially, when there is no working copy configured the **All Files** view mode lists the following two actions:



**Figure 388: All Files Panel**

For **Modified**, **Incoming**, **Outgoing**, **Conflicts** view modes, the following actions may be available, depending on the current working copy state in different contexts:

-  **Information message** - Informs you why there are no resources presented in the currently selected view mode.
-  **Synchronize with Repository** - Available only when there is nothing to present in the **Modified** and **Incoming** view modes.
-  **Switch to Incoming** - Selects the **Incoming** view mode.
-  **Switch to Outgoing** - Selects the **Outgoing** view mode.
-  **Switch to Conflicts** - Selects the **Conflicts** view mode.
-  **Show all changes/incoming/outgoing/conflicts** - Depending on the currently selected view mode, this action presents the corresponding resources after a synchronize operation was executed only on a part of the working copy resources.

## History View

In Apache Subversion™, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from [Repositories view](#), [Working Copy view](#), [Revision Graph](#), or [Directory Change Set view](#). From the **Working copy view** you can display the history of local versioned resources.

The view consists of four distinct areas:

- The table showing details about each revision, like: revision number, commit date and time, number of changes (more details available in the tooltip), author's name, and a fragment of the commit message.

Some revisions may be highlighted to emphasize:

- the current revision of the resource for which the history is displayed - a bold font revision.
- the last revision in which the content or properties of the resource were modified - blue font revision.



**Note:** Both font highlights may be applied for the same revision.

- The complete commit message for the selected revision.
- A tree structure showing the folders where the modified resources are located. You can compress this structure to a more compact form that focuses on the folders that contain the actual modifications.
- The list of resources modified in the selected revision. For each resource, the type of action done against it is marked with one of the following symbols:
  - - A newly created resource.
  - - A newly created resource, copied from another repository location.
  - - The content/properties of the resource were *modified*.
  - - Resource was *replaced* in the repository.
  - - Resource was deleted from the repository.

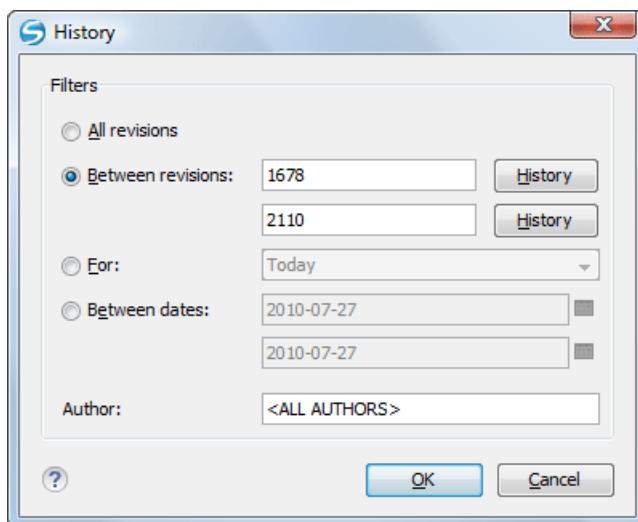
The screenshot shows the History View window for the file "site.xml". The toolbar at the top includes icons for Group by date, Refresh, and a search bar labeled "Type filter text". The main area displays a table of revisions grouped by time frame: Today (1 revision), Last week (1 revision), and Last month (35 revisions). The "Last month" section is expanded, showing 35 revisions from October 2011. The selected revision (highlighted in blue) is 16244, dated 2011-10-24 17:32:24, made by user george, with the message "More updates on new features.". Below the table, a note says "More updates on new features." A sidebar on the left shows the project structure under "http://devel-site.sync.ro/svn/repos": www.oxygenxml.com/trunk/xml/site.xml. The right side shows a table for copied files, listing "site.xml" with a path of "/www.oxygenxml.com/trunk/xml".

**Figure 389: History View**

You can group revisions in predefined time frames (today, yesterday, this week, this month), by pressing the **Group by date** button from the toolbar.

#### The History Filter Dialog Box

The **History view** does not always show all the changes ever made to a resource because there may be thousands of changes and retrieving the entire list can take a long time. Normally you are interested in the more recent ones. That is why you can specify the criteria for the revisions displayed in the **History view** by selecting one of several options presented in the **History** dialog box that is displayed when you invoke the **Show History** action.



**Figure 390: History Filters Dialog Box**

Options for the set of revisions presented in the History view are:

- All revisions of the selected resource.
- Only revisions between a start revision number and an end revision number.
- Only revisions added in a period of time like today, last week, last month, etc.
- Only revisions between a start and an end date.
- Only revisions committed by a specified SVN user.

The toolbar of the **History view** has two buttons for extending the set of revisions presented in the view: **Get next 50** and **Get all**.

### **The History Filter Field**

When only the history entries which contain a specified substring need to be displayed in the **History view** the filter field displayed at the top of this view is the perfect fit. Just enter the search string in the field next to the label **Find**. Only the items with an author name, commit message, revision number or date which match the search string are kept in the **History view**. The filter action is executed and the content of the table is updated when the button  **Search** is pressed.

### **Contextual Menu Actions**

The **History view** contains the following contextual menu actions:

#### **Compare with working copy**

Compares the selected revision with your working copy file. It is enabled only when you select a file.

#### **Open**

Opens the selected revision of the file into the Editor. This is enabled only for files.

#### **Open with**

Displays the **Open with...** dialog box to specify the editor in which the selected file will be opened.

#### **Get Contents**

Replaces the current version from the working copy with the contents of the selected revision from the history of the file. The *BASE* version of the file is not changed in the working copy so that after this action the file will appear as modified in a synchronization operation, that is newer than the *BASE* version, even if the contents is from an older version from history.

#### **Save as**

Allows you to save the contents of a file as it was committed at a certain revision. This option is available only when you access the history of a file.

#### **Copy to**

Copies to the repository the item whose history is displayed, using the selected revision. This option is active only when presenting the history for a repository item (URL).

 **Note:** This action can be used to resurrect deleted items also.

#### **Revert changes from this revision**

Reverts changes that were made in the selected revisions. The changes are reverted only in your working copy and does not affect the repository items. It does not replace your working copy items with those from the selected revisions. This action is enabled when the resource history was launched for a local working copy resource.



**Note:** For items displayed in the **Affected Paths** section that were *added*, *deleted*, or *replaced*, this action has no effect because such changes are considered to be changes to the parent directory. To revert these type of changes, follow these steps:

1. Request the history for the parent directory.
2. Identify the revision that contains the changes you want to revert.
3. Invoke the action on that revision.



**Warning:** There are instances where the SVN Client is not able to identify the corresponding working copy item for the selected item in the **Affected Paths** section. In this case, the action does not proceed and an error message is displayed. For example, the selected item in the **Affected Paths** section is from a different repository location than the working copy item for which the history is displayed.

### Update to revision

Updates your working copy resource to the selected revision. This is useful if you want your working copy to reflect a time in the past. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy is inconsistent and you are unable to commit your changes.

### Check out

Checks out a new working copy of the directory for which the history is presented, from the selected revision.

### Export...

Opens [the Export dialog box](#) that allows you to configure options for exporting a folder from the repository to the local file system.

### Show Annotation... ([Ctrl Shift A \(Command Shift A on OS X\)](#))

Opens the **Show Annotation** dialog box that computes [the annotations for a file and displays them in the Annotations view](#), along with the history of the file in the **History** view.

### Change

Allows you to change commit data for a file:

- *Author* - Changes the name of the SVN user that committed the selected revision.
- *Message* - Changes the commit message of the selected revision.

When two resources are selected in the **History** view, the contextual menu contains the following actions:

### Compare revisions

When the resource is a file, the action compares the two selected revisions using the **Compare** view. When the resource is a folder, the action displays the set of all resources from that folder that were changed between the two revision numbers.

### Revert changes from these revisions

Similar to the `svn merge` command, it merges two selected revisions into the working copy resource. This action is only enabled when the resource history was requested for a working copy item.

For more information about the **History view** and its features please read the sections [Request history for a resource](#) and [Using the resource history view](#)

### Directory Change Set View

The result of comparing two reference revisions from the history of a folder resource is a set with all the resources changed between the two revision numbers. The changed resources can be contained in the folder or in a subfolder of that folder. These resources are presented in a tree format. For each changed resource all the revisions committed between the two reference revision numbers are presented.

Action	Revision	Date	Author	Message
Modified	10820	2010-07-09 15:48:19	sorin	EXM-17248...
Modified	10716	2010-07-01 14:54:07	sorin	EXM-17949...
Modified	10527	2010-06-14 14:53:30	sorin	EXM-1684...
Modified	10125	2010-04-20 17:17:46	sorin	EXM-16856...
Modified	10093	2010-04-13 15:46:10	bogdan	Reviewed.
Modified	10088	2010-04-13 14:09:38	sorin	EXM-16849...
Modified	10076	2010-04-09 14:19:22	sorin	EXM-16856...
Modified	10075	2010-04-09 13:04:44	sorin	EXM-17248...
Modified	10074	2010-04-09 12:31:56	bogdan	Reviewed.
Modified	10072	2010-04-09 12:04:13	sorin	EXM-17248...

Commit message  
EXM-17248 Display child map AuthorDevelGuide as one entry in parent map  
EditorUserManual.ditamap.

**Figure 391: Directory Change Set View**

The set of changed resources displayed in the tree is obtained by running the action **Compare revisions** available on the context menu of the **History** view when two revisions of a folder resource are selected in the **History** view.

The left side panel of the view contains the tree hierarchy with the names of all the changed resources between the two reference revision numbers. The right side panel presents the list with all the revisions of the resource selected in the left side tree. These revisions were committed between the two reference revision numbers. Selecting one revision in the list displays the commit message of that revision in the bottom area of the right side panel.

A double click on a file listed in the left side tree performs a diff operation between the two revisions of the file corresponding to the two reference revisions. A double click on one of the revisions displayed in the right side list of the view performs a diff operation between that revision and the previous one of the same file.

The contextual menu of the right side list contains the following actions:

#### Compare with previous version

Performs a diff operation between the selected revision in the list and the previous one.

#### Open

Opens the selected revision in the associated editor type.

#### Open with

Displays a dialog with the available editor types and allows the user to select the editor type for opening the selected revision.

#### Save as

Saves the selected file as it was in the selected revision.

#### Copy to

Copies to the repository the item whose history is displayed, using the selected revision.

**Note:** This action can be used to resurrect deleted items also.

#### Check out

Checks out a new working copy of the selected directory, from the selected revision.

#### Export...

Opens *the Export dialog box* that allows you to configure options for exporting a folder from the repository to the local file system.

 **Show Annotation... (Ctrl Shift A (Command Shift A on OS X))**

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the Annotations view*, along with the history of the file in the **History** view.

 **Show SVN Information (Ctrl (Command on OS X) + I)**

Provides additional information for a selected resource. For more details, go to [Obtain information for a resource](#).

### The Editor Panel of SVN Client

You can open a file for editing in an internal built-in editor. There are default associations between frequently used file types and the internal editors in *the File Types preferences panel*.

The internal editor can be accessed either from the *Working copy view* or from the *History view*. No actions that modify the content are allowed when the editor is opened with a revision from history.

Only one file at a time can be edited in an internal editor. If you try to open another file it will be opened in the same editor window. The editor provides syntax highlighting for known file types. This means that a different color will be used for each recognized token type found in the file. If the file's content type is unknown you will be prompted to choose the proper way the file should be opened.

After editing the content of the file in an internal editor you can save it to disk by using the **Save** action from the *File* menu or the **Ctrl S (Command S on OS X)** key shortcut. After saving your file you can see the file changed status in *the Working Copy view*.

If the internal editor associated with a file type is not the XML Editor, then the encoding set in *the preference Encoding for non XML files* is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the file's encoding.

### Annotations View

Sometimes you need to know not only what was changed in a file, but also who made those changes. This view displays the revision and the author that changed every line in a file. Just click on a line in the editor panel where the file is opened to see the revision in which the line was last modified. The same revision is highlighted in the **History view** and you can also see all the lines that were changed in the same revision highlighted in the editor panel. Also, the entries of the **Annotations view** corresponding to that revision are highlighted. Therefore, the **Annotations view**, **History view**, and annotations editor panel are all synchronized. Clicking on a line in one of them highlights the corresponding lines in the other two.

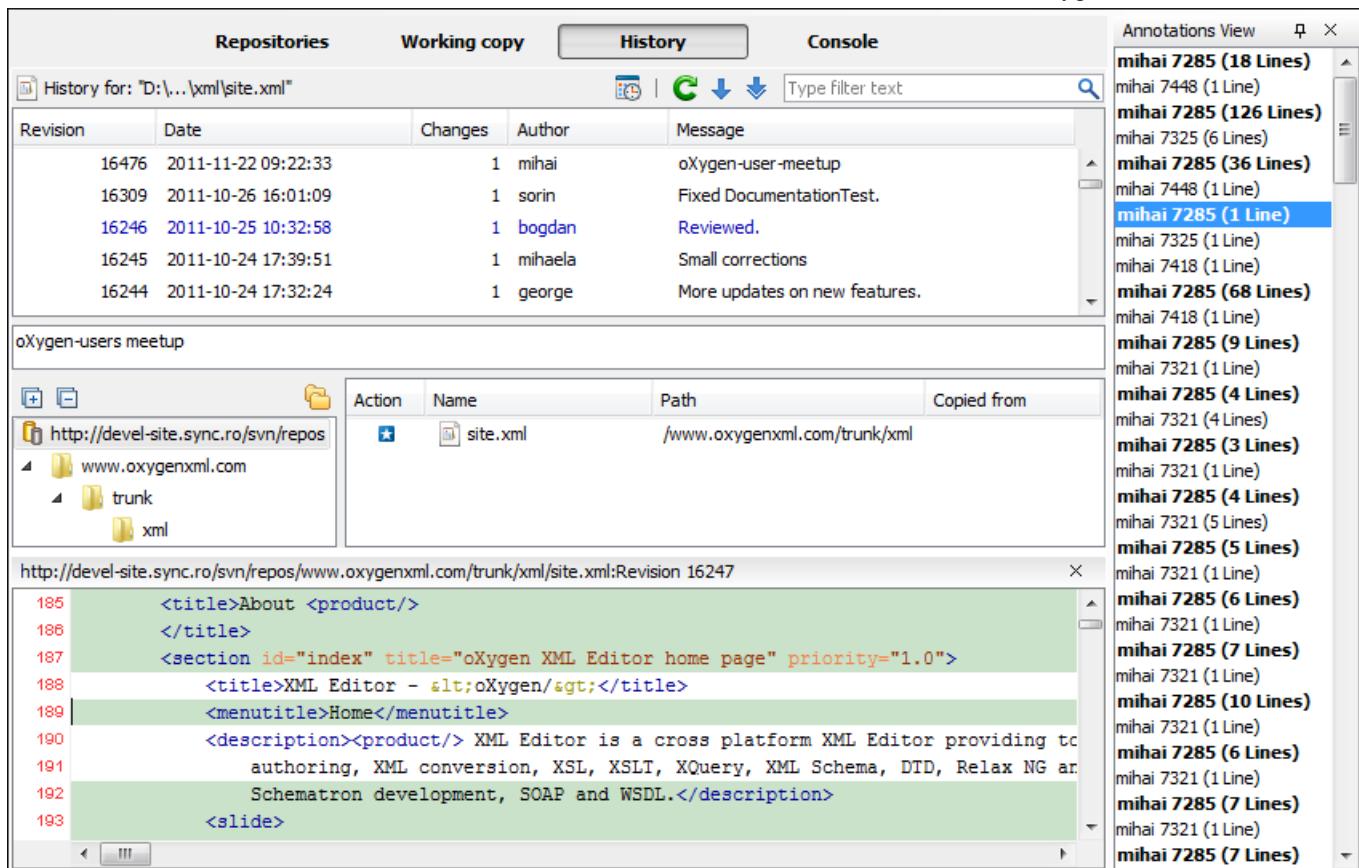
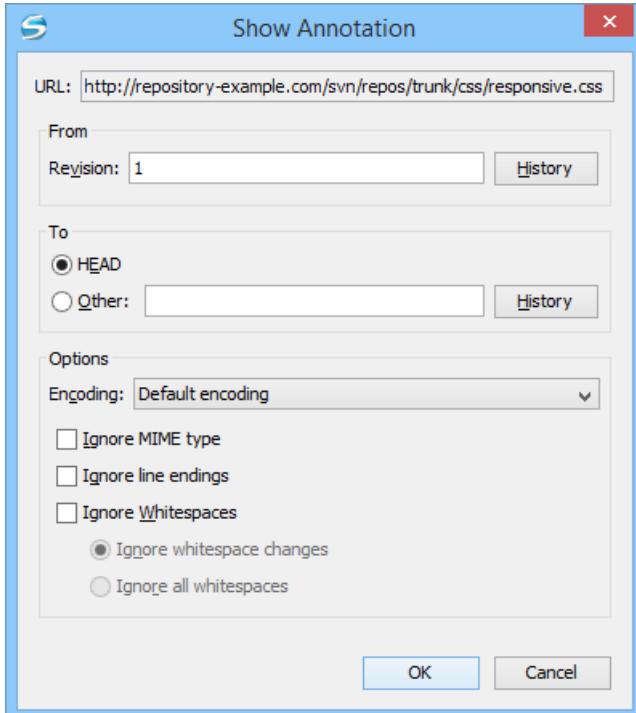


Figure 392: The Annotations View

The annotations of a file are computed with the **Show Annotation...** action, which is available in the **History** menu, and from the contextual menu of the following views: *the Repositories view, Working copy view, History view, and Directory Change Set view*.



**Figure 393: The Show Annotation Options Dialog Box**

The following options can be configured in the **Show Annotation** dialog box:

#### From Revision Section

Select the revision from which to start computing the annotation. If you press the **History** button, *the History dialog box* is displayed, which allows you to select a revision.

#### To Revision Section

Select the ending revision by choosing between the **HEAD** revision or specify it in the **Other** text box . If you press the **History** button, *the History dialog box* is displayed, which allows you to select a revision.

#### Encoding

Select the encoding to be used when the annotation is computed. For each line of text, the SVN Client looks through the history of the file to be annotated see when it was last modified, and by whom. It is required that it is in the form of a text file. Therefore, encoding is needed to properly decode and read the file content. By default, the encoding of the operating system is used.

#### Ignore MIME type

If enabled, the file is treated as a text file and ignores what the SVN system infers from the `svn:mime-type` property.

#### Ignore line endings

If enabled, the differences in line endings are ignored when the annotation is computed.

#### Ignore whitespaces

If enabled, it allows you to specify how the whitespace changes should be handled. When enabled, you can then choose between two options:

- **Ignore whitespace changes** - Ignores changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces).



**Note:** Whitespace that were added where there were none before, or that were removed, are still considered to be changes.

- **Ignore all whitespaces** - Ignores all types of whitespace changes.



**Tip:** Enabling any of these *ignore* options can help you better determine the last time a meaningful change was made to a given line of text.

After you configure the options and press **OK**, the annotations will be computed and the **Annotations** view is displayed, where all the users that modified the selected resource will be presented, along with the specific lines and revision numbers modified by each user.



**Note:** If the file has a very long history, the computation of the annotation data can take a long time to process.

## Compare View

In the Oxygen XML Editor there are three types of files that can be checked for differences: text files, image files and binary files. For the text files and image files you can use the built-in **Compare** view.

```
E:\NewSamples\personal.css*
personal.css
11 border-bottom: 0.1em solid navy;
12 padding: 0.3em;
13
14 font-size:large;
15 font-weight:bold;
16
17 color:navy;
18 background-color:inherit;
19
20 }
21
22 personnel{
23 display:block;
24 margin:1em;
25 /*Counter for the person elements*/
26 counter-reset:pers_cnt;
27 }
28
29 person{
30 display:block;
31
32 margin: 1em;
33 font-size:medium;
34 font-weight:normal;
35 border:0.1em solid #DDDDEE;
36 padding:0.5em;
37
38 color:inherit;
39 background-color: #EFEFEE;
40 }

personal.css@HEAD [dragos]
}
personnel:before{
 display:block;
 content:"List of employees";
}

border-bottom: 0.4em solid navy;
padding: 0.2em;
font-size:small;
font-weight:bold;
color:blue;
background-color:inherit;
}

person{
 display:block;
}

margin: 2em;
border:0.3em solid #DDDDEE;
padding:0.2em;
color:inherit;
background-color: #EFECCC;

/*Increments the person counter.*/
counter-increment:pers_cnt;

name,
family,
given,
```

**Figure 394: Compare View**

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

When comparing text, the differences are computed using a *line differencing algorithm*. The view can be used to show the differences between two files in the following cases:

- after obtaining the outgoing status of a file with a **Refresh** operation, the view can be used to show the differences between your working file and the pristine copy. In this way you can find out what changes you will be committing;
- after obtaining the incoming and outgoing status of the file with the **Synchronize** operation, you can examine the exact differences between your local file and the **HEAD** revision file;

- you can use the **Compare view** from the **History view** to compare the local file and a selected revision or compare two revisions of the same file.

The Compare view contains two editors. Edits are allowed only in the left editor and only when it contains the working copy file. To learn more about how the view can be used in the day by day work see [View differences](#).

### Compare View Toolbar

The list of actions available in the **Compare** view toolbar include:



#### Save action

Saves the content of the left editor when it can be edited.



#### Perform Files Differencing

Performs a comparison between the source file and target file.



#### Ignore Whitespaces

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.



#### Synchronized scrolling

Synchronizes scrolling so that a selected difference can be seen on both sides of the application window. This action enables/disables the previously described behavior.



#### Format and Indent Both Files ([Ctrl Shift P \(Command Shift P on OS X\)](#))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.



#### Next Block of Changes ([Ctrl . \(Command . on OS X\)](#))

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes.



**Note:** A change block groups one or more consecutive lines that contain at least one change.



#### Copy Change from Right to Left

Copies the selected difference from the target file in the right side to the source file in the left side.



#### Copy All Changes from Right to Left

Copies all changes from the target file in the right side to the source file in the left side.



#### Previous Block of Changes ([Ctrl , \(Command , on OS X\)](#))

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes.



#### Next Change ([Ctrl Shift . \(Command Shift . on OS X\)](#))

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change or when there are no changes.



#### Previous Change ([Ctrl Shift , \(Command Shift , on OS X\)](#))

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change or when there are no changes.



#### First Change ([Ctrl B \(Command B on OS X\)](#))

Jumps to the first change.

Most of these actions are also available from the [Compare](#) menu.

## Image Preview

You can view your local files by using the built-in **Image preview** component. The view can be accessed from the [Working copy view](#) or from the [Repository view](#). It can also be used from the [History view](#) to view a selected revision of a image file.

Only one image file can be opened at a time. If an image file is opened in the *Image preview* and you try to open another one it will be opened in the same window. Supported image types are *GIF, JPEG/JPG, PNG, BMP*. Once the image is displayed in the **Image preview** panel using the actions from the contextual menu one can scale the image at its original size (**1:1** action) or scale it down to fit in the view's available area (**Scale to fit** action).

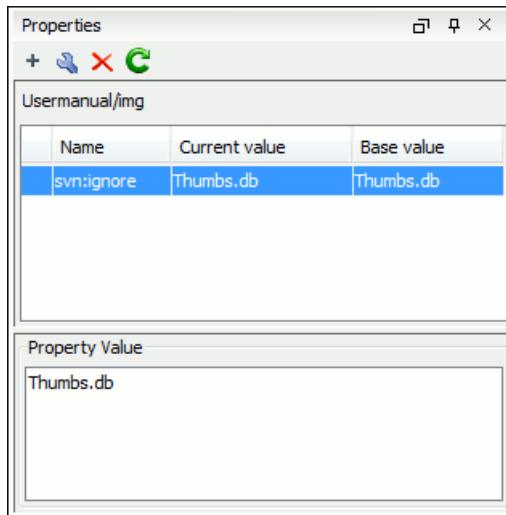
## Compare Images View

The images are compared using the Compare images view. The images are presented in the left and right part of the view, scaled to fit the view's available area. You can use the contextual menu actions to scale the images at their original size or scale them down to fit the view's available area.

The supported image types are: *GIF, JPG / JPEG, PNG, BMP*.

## Properties View

The properties view presents Apache Subversion™ properties for the currently selected resource from either the **Working Copy** view or the **Repositories** view.



**Figure 395: The Properties View**

Above the table it is specified the currently active resource for which the properties are presented. Here you will also find a warning when an unversioned resource is selected.

The table in which the properties are presented has four columns:

- **State** - can be one of:
  - (empty) - normal unmodified property, same current and base values;
  - \*(asterisk) - modified property, current and base values are different;
  - +(plus sign) - new property;
  - -(minus sign) - removed property.
- **Name** - the property name.
- **Current value** - the current value of the property.
- **Base value** - the base(original) value of the property.

## The `svn:externals` Property

The `svn:externals` property can be set on a folder or a file. In the first case it stores *the URL of a folder from other repository*.

In the second case it stores the URL of a file from other repository. The external file will be added into the working copy as a versioned item. There are a few differences between directory and file externals:

- The path to the file external must be in a working copy that is already checked out. While directory externals can place the external directory at any depth and it will create any intermediate directories, file externals must be placed into a working copy that is already checked out.
- The external file URL must be in the same repository as the URL that the file external will be inserted into; inter-repository file externals are not supported.
- While commits do not descend into a directory external, a commit in a directory containing a file external will commit any modifications to the file external.

The differences between a normal versioned file and a file external:

- File externals cannot be moved or deleted; the `svn:externals` property must be modified instead; however, file externals can be copied.

A file external shows up as a X in the switched status column.

## Toolbar / Contextual Menu

The properties view toolbar and contextual menu contain the following actions:

-  **Add a new property** - This button invokes the *Add property* dialog in which you can specify the property name and value.
-  **Edit property** - This button invokes the *Edit property* dialog in which you can change the property value and also see its original(base) value.
-  **Remove property** - This button will prompt a dialog to confirm the property deletion. You can also specify if you want to remove the property recursively.
-  **Refresh** - This action will refresh the properties for the current resource.

## Console View

The **Console View** shows the traces of all the actions performed by the application. Part of the displayed messages mirror the communication between the application and the Apache Subversion™ server. The output is expressed as subcommands to the Subversion server and simulates the Subversion command-line notation. For a detailed description of the Subversion console output read the [SVN User Manual](#).

The view has a simple layout, with most of its space occupied by a message area. On its right side, there is a toolbar holding the following buttons:

### **Clear**

Erases all the displayed messages.

### **Lock scroll**

Disables the automatic scrolling when new messages are appended in the view.

The maximum number of lines displayed in the console (length of the buffer) can be modified in the [Preferences](#) page. By default this value is set to 100.

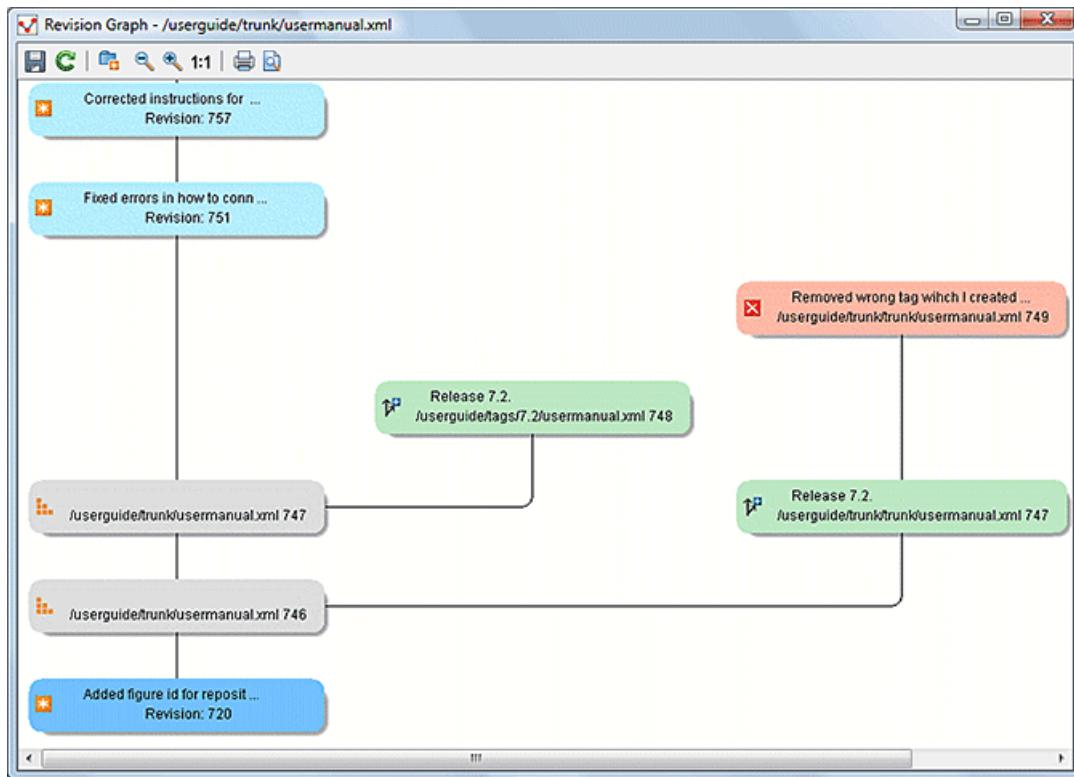
## Dynamic Help View

**Dynamic Help view** is a help window that changes its content to display the help section that is specific to the currently selected view. As you change the focused view, you are able to read a short description of it and its functionality.

## The Revision Graph of a SVN Resource

The history of a SVN resource can be watched on a graphical representation of all the revisions of that resource together with the tags in which the resource was included. The graphical representation is identical to a tree structure and very easy to follow.

The graphical representation of a resource history is invoked with the  **Revision graph** action available on the right click menu of a SVN resource in *the Working Copy view* and *the Repository view*.



**Figure 396: The Revision Graph of a File Resource**

In every node of the revision graph an icon and the background color represent the type of operation that created the revision represented in that node. Also the commit message associated with that revision, the repository path and the revision number are contained in the node. The tooltip displayed when the mouse pointer hovers over a node specifies the URL of the resource, the SVN user who created the revision of that node, the revision number, the date of creation, the commit message, the modification type and *the affected paths*.

The types of nodes used in the graph are:

### Added resource

The  icon for a new resource added to the repository and a green background.

### Copied resource

The  icon for a resource copied to other location, for example when a SVN tag is created and a green background.

### Modified resource

The  icon for a modified resource and a blue background.

### Deleted resource

The  icon for a resource deleted from the repository and a red background.

### Replaced resource

The  icon for a resource removed and replaced with another one on the repository and a orange background.

## Indirect resource

The  icon for a revision from where the resource was copied or an indirectly modified resource, that is a directory in which a resource was modified and a grey background. The *Modification type* field of the tooltip specifies how that revision was obtained in the history of the resource.

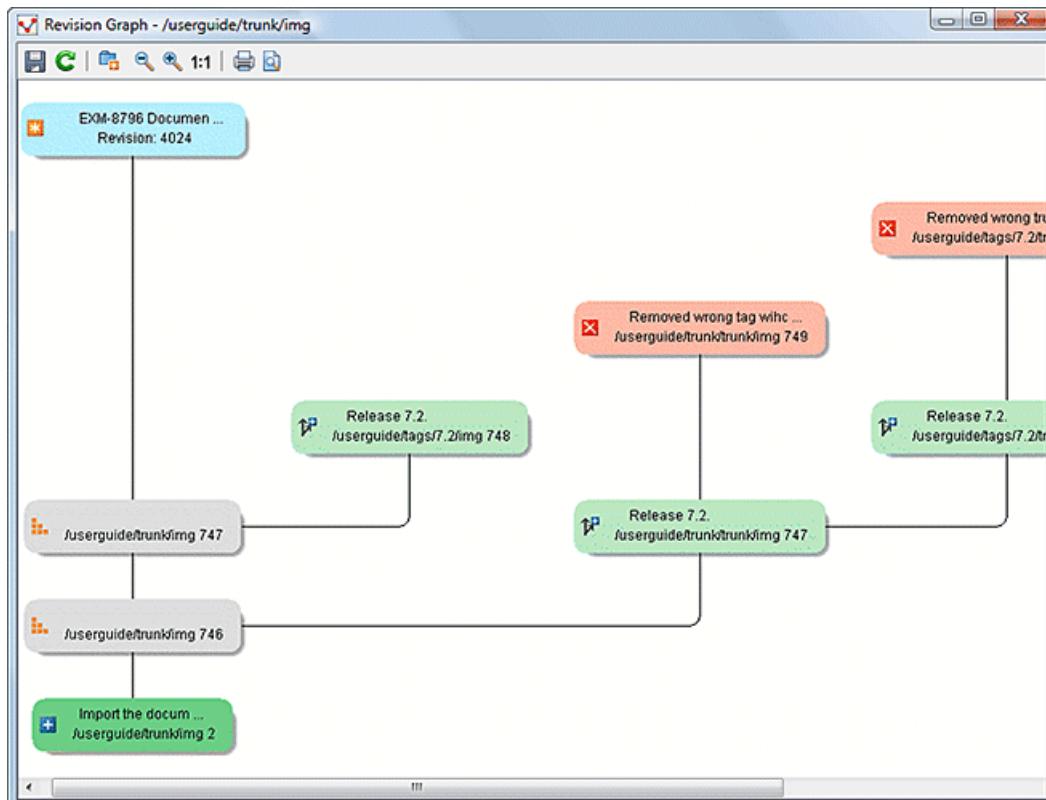
A directory resource is represented with two types of graphs:

### simplified graph

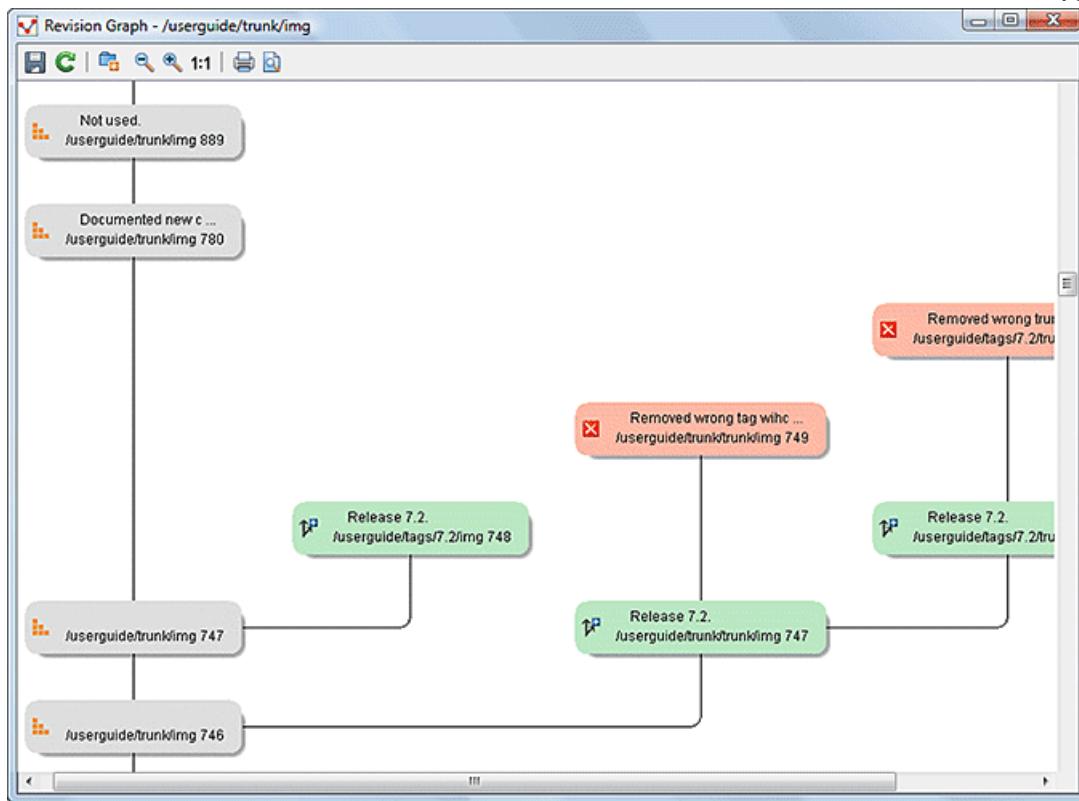
Lists only the changes applied directly to the directory;

### complete graph

Lists also the indirect changes of the directory resource, that is the changes applied to the resources contained in the directory.



**Figure 397: The Revision Graph of a Directory (Direct Changes)**



**Figure 398: The Revision Graph of a Directory (Also Indirect Changes)**

The **Revision graph** dialog toolbar contains the following actions:

**Save as image**

Saves the graphical representation as image. For a large revision graph you have to [set more memory in the startup script](#). The default memory size is not enough when there are more than 100 revisions that are included in the graph.

**Show/Hide indirect modifications**

Switches between simplified and complete graph.

**Zoom In**

Zooms in the graph.

**Zoom Out**

Zooms out the graph. When the font reaches its minimum size, the graph nodes will display only the icons, leading to a very compact representation of the graph.

**1:1 Reset scale**

Resets the graphical scale to a default setting.

**Print**

Prints the graph.

**Print preview**

Offers a preview of the graph to allow you to check the information to be printed.

The contextual menu of any of the graph nodes contains the following actions:

**Open**

Opens the selected revision in the editor panel. Available only for files.

**Open with**

Opens the selected revision in the editor panel. Available only for files.

**Save as**

Saves the file for which the revision graph was generated, based on the selected node revision.

**Copy to**

Copies to the repository the item whose revision graph is displayed, using the selected revision.



**Note:** This action can be used to resurrect deleted items also.

**Compare with HEAD**

Compares the selected revision with the HEAD revision and displays the result in the diff panel. Available only for files.

**Show History**

Displays the history of the resource in [the History view](#). Available for both files and directories.

**Check out**

[Checks out](#) the selected revision of the directory. Available only for directories.

**Export...**

Opens [the Export dialog box](#) that allows you to configure options for exporting a folder from the repository to the local file system.

When two nodes are selected in the revision graph of a file the right click menu of this selection contains only the **Compare** for comparing the two revisions corresponding to the selected nodes. If the resource for which the revision graph was built is a folder then the right click menu displayed for a two nodes selection also contains the **Compare** action but it computes the differences between the two selected revisions as a set of directory changes. The result is displayed in the [Directory Change Set](#) view.

**Attention:**

Generating the revision graph of a resource with many revisions may be a slow operation. You should enable caching for revision graph actions so that future actions on the same repository will not request the same data again from the SVN server which will finish the operation much faster.

## Oxygen XML Editor SVN Preferences

The options used in the SVN client are saved and loaded independently from the Oxygen XML Editor options. However, if Oxygen XML Editor cannot determine a set of SVN options to be loaded at startup, some of the preferences are imported from the XML Editor options (such as the License key and HTTP Proxy settings).

There is also an additional set of preferences applied to the SVN client that are set in global SVN files. There are two editing actions available in the **Global Runtime Configuration** submenu of the **Options** menu. These actions, **Edit 'config' file** and **Edit 'servers' file**, contain parameters that act as defaults applied to all the SVN client tools that are used by the same user on their login account.

## Entering Local Paths and URLs

The Oxygen XML Editor includes a variety of option configuration pages or wizards that contain text boxes where you specify paths to local resources or URLs of items inside remote repositories. The Oxygen XML Editor provides support in these text boxes to make it easier to specify these paths and URLs.

### Local Item Paths

The text boxes used for specifying local item paths support the following:

- *Absolute Paths* - In most cases, the Oxygen XML Editor expects absolute paths for local file system items.
- *Relative Paths* - The Oxygen XML Editor only accepts relative paths in the form ~[ / . . . ], where ~ is the user home directory.
- *Path Validation* - Oxygen XML Editor validates the path as you type and invalid text becomes red.

- *Drag and Drop* - You can drag files and folders from the file system or other applications and drop them into the text box.
- *Automatic Use of Clipboard Data* - If the text box is empty when its dialog box is opened, any data that is available in the system clipboard is used as long as it is valid for that text box.

## Repository Item URLs

- *Local Repository Paths* - You can use local paths (absolute or relative) to access local repositories. When you use the **Browse** button, the Oxygen XML Editor will convert the file path to a `file://` form of URL as long as the location is a real repository.
  - *Absolute Paths* - In most cases, the Oxygen XML Editor expects absolute paths for local file system items.
  - *Relative Paths* - The Oxygen XML Editor only accepts relative paths in the form `~[ / . . . ]`, where `~` is the user home directory.
- *Peg Revisions* - For URL text boxes found inside dialog boxes where you are pulling information from the repository, you can *use peg revisions at the end of the URLs* (for example, `URL@rev1234`).
  -  **Note:** If you try to use a *peg revision* number in a dialog box where you are sending information to the repository then the peg revision number will become part of the name of the item rather than searching for the specified revision. For example, in the URL `http://host/path/inside/repo/item@100`, the item name is considered to be `item@100`.
  -  **Tip:** You can even use *peg revisions* with local repository paths. For example, `C:\path\to\local\repo@100` will be converted to `file:///C:/path/to/local/repo@100` and the **Repository browser** will display the content of the local repository as it is at revision 100.

- *URL Validation* - Oxygen XML Editor validates the URLs as you type and invalid text becomes red. Even paths to local repositories are not accepted unless using the **Browse** button to convert them to valid URLs.
- *Drag and Drop* - You can drag URLs from other applications or text editors and drop them into the URL text box. You can also drag folders that point to local repositories, from the local file system or from other applications, and they are automatically converted to valid `file://` type URLs.
- *Automatic Use of Clipboard Data* - If the URL text box is empty when its dialog box is opened, any data that is available in the system clipboard is used as long as it is valid for that text box. Even valid local paths will be automatically converted to `file://` type URLs.

-  **Note:** The text boxes that are in the form of a combo box also allow you to select previously used URLs, or URLs defined in the **Repositories** view.

## Technical Issues

This section contains special technical issues found during the use of Syncro SVN Client.

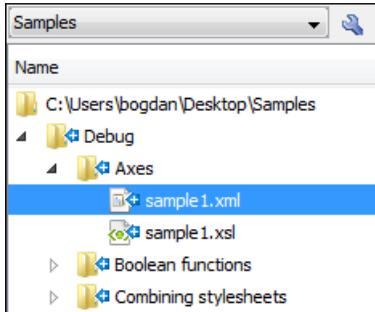
### Authentication Certificates Not Saved

If Syncro SVN Client prompts you to enter the authentication certificate, although you already provided it in a previous session, then you should make sure that your local machine user account has the necessary rights to store certificate files in the *Subversion* configuration folder (write access to *Subversion* folder and all its subfolders). Usually, it is located in the following locations:

- Windows: `[HOME_DIR]\AppData\Roaming\Subversion`
- Mac OS X and Linux: `[HOME_DIR]/.subversion`

### Updating Newly Added Resources

When you want to get from the repository a resource which is part of a newly created structure of folders, you need to also get its parent folders.



**Figure 399: An incoming structure of folders from the repository**

Syncro SVN Client allows you to choose how you want to deal with the entire structure from that moment onwards:

#### Update ancestor directories recursively

This option brings the entire newly added folders structure into your working copy. In this case, the update time depends on the total number of newly incoming resources, because of the full update operation (not updating only selected resource).

#### Update selected files only (leave ancestor directories empty)

This option brings a skeleton structure composed of the resource's parent folders only, and the selected resource at the end of the operation. All of the parent directories will have depth set to *empty* in your working copy, thus subsequent **Synchronize** operations will not report any remote modifications in those folders. If you need to update the folders to full-depth, you can use **Update to revision/depth** option from the working copy.

### Cannot Access a Repository through HTTPS

If you have issues when trying to access a repository through HTTPS protocol, one of the possible causes can be the encryption protocol currently used by the application. This is happening when:

- you are running Oxygen XML Editor with Java 1.6 or older.
- the repository is set to use only one of the SSLv3 or TLSv1 encryption protocols.

To solve this issue, set the **HTTPS encryption protocols** option to **SSLv3 only** or **TLSv1 only** (depending on the repository configuration).

### Accessing Old Items from a Repository

Usually, you point to an item from a repository using a URL. However, sometimes this might not be enough, because the URL alone might point to a different item than the one you want and a *peg revision* is needed.

A Subversion repository tracks any change made to its items by using *revisions*, which contain information like the name of the author who made the changes, the date when they were made, and a number that uniquely identifies each of them. During time, an item from a specific location in a repository evolves as a result of committing different changes to it. When an item is deleted, its entire life cycle (all changes made to it, from the moment it was created) still remains recorded in the history of the repository. If a new item is created, with the same name and in the same location of the repository as a previously existing one, then both items are identified by the same URL even though they are located in different time frames. This is when a *peg revision* comes in handy. A *peg revision* is nothing more than a normal revision, but the difference between them is made by their usage. Many of the Subversion commands accept also a peg revision as a way to precisely identify an item in time, beside an *operative revision* (the revision we are interested in, regarding the used command).

Let's assume that:

- we created a new repository file `config`, identified by the URL `http://host.com/myRepository/dir/config`.
- the file has been created at revision 10.
- during time, the file was modified by committing revisions 12, 15, 17.

To access a specific version of the file identified by the `http://host.com/myRepository/dir/config` URL, we need to use a corresponding revision (the operative revision):

- if we use a revision number less than 10, an error is triggered, because the file has not been created yet.
- if we use a revision number between 10 and 19, we will obtain the specific version we are interested in.



**Note:** Although the file was modified in revisions 12, 15, 17, it existed in all revisions between 10 and 19. Starting with a revision at which the file is modified, it has the same content across all revisions generated in the repository until another revision in which it is modified again.

At this point, we delete the file, creating revision 20. Now, we cannot access any version of the file, because it does not exist anymore in the latest repository revision. This is due to the fact that Subversion automatically uses the HEAD revision as a peg revision (it assumes any item currently exists in the repository if not instructed otherwise). However, using any of the revision numbers from the 10–19 interval (when the file existed) as a peg revision (beside the operative revision), will help Subversion to properly identify the time frame when the file existed, and access the file version corresponding to the operative revision. If we use a revision number greater than 19, this will also trigger an error.

Continuing our example, let's suppose that at revision 30 we create a directory, incidentally called `config`, in the same repository location as our deleted file. This means that our new directory will be identified by the same repository address: `http://host.com/myRepository/dir/config`. If we use only this URL in any Subversion command, we will access the new directory. We will also access the same directory if we use as peg revision any revision number equal with or greater than 30. But, if we are interested in accessing an old version of the previously existing file, then we must use as a peg revision one of the revisions at which it existed (10–19), just like in the previous case.

## Checksum Mismatch Error

A *Checksum Mismatch* error could happen if an operation that sends or retrieves information from the repository to the working copy is interrupted. This means that there is a problem with the synchronization between a local item and its corresponding remote item.

If you encounter this error, try the following:

1. Identify the parent directory of the file that caused the error (the file name should be displayed in the error message).



**Note:** If the parent directory is the root of the working copy or if it contains a large amount of items it is recommended that you check out the working copy again, rather than continuing with the rest of this procedure.

2. Identify the *current depth* of that directory.

3. Update the parent directory using the **Update to revision/depth** action that is available from the contextual menu or the **Working copy** menu.

- a. For the **Depth** option, select **This folder only (empty)**.



**Warning:** If you have files with changes in this directory, those changes could be lost. You should commit your changes or move the files to another directory outside the working copy prior to proceeding with this operation.

4. After clicking **OK** the contents of the directory will be erased and the directory is be marked as having *an empty depth*.

5. Once again, update the same directory using the **Update to revision/depth** action.

- a. This time, for the **Depth** option, select the depth that was previously identified in step 2.

6. If you moved modified files to another directory outside the working copy, move them back to the original location inside the working copy.

If this procedure does not solve the error, you need to check out the working copy again and move possible changes from the old working copy to the new one.

## Tree Editor

---

The **Tree Editor** (**Tools > Tree Editor**) is used for editing the content of a document displayed as an XML tree. The workspace offers the following functional areas:

- Main menu - provides access to all the features and functions available in Oxygen XML Editor Tree Editor perspective;
- Toolbar - provides easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function;
- Editor panel - easy editing of structured mark-up documents. Each token has an associated icon for easier visual identification;
- Message panel - displays messages returned from user operations;
- Model view - shows the detailed information about the attribute or element that you are working on;
- All Elements panel - presents a list of all defined elements that can be inserted within your document.

The tree editor does not offer entity support. Entities are not presented with a special type of node in the tree and new entity nodes cannot be inserted in the document.

## Comparing and Merging Documents

---

In large teams composed of developers or technical writers, the use of a shared repository for the source or document files is a must. Often times multiple authors may be editing the same file at the same time. It is easy to manage multiple changes by using the Oxygen XML Editor comparison and merging tools.

It can be difficult to recognize which files and folders have been modified. If your data has changed, you can use the helpful Oxygen XML Editor features for comparing files and directories to accurately identify and process changes in your files and folders. These tools are powerful, easy-to-use, and produce fast, thorough results.

Oxygen XML Editor provides a simple means of performing file and folder comparisons. You can see the differences in your files and folders and merge the changes.

There are two types of comparison tools: **Compare Directories** or **Compare Files**. These utilities are available from the **Tools** menu or can be opened as stand-alone applications from the Oxygen XML Editor installation folder (`diffDirs.exe` and `diffFiles.exe`).

 **Note:** File comparison and merging actions can also be performed on files inside ZIP-based archives.

The **Compare Files** tool can also be used to compare XML fragments. They can be compared and merged by copying and pasting the fragments into both sides of the comparison window, without selecting files.

The comparison tools can also be started by using command-line arguments. In the installation folder there are two executable shells (`diffFiles.bat` and `diffDirs.bat` on Windows, `diffFiles.sh` and `diffDirs.sh` on Unix/Linux, `diffFilesMac.sh` and `diffDirsMac.sh` on OS X). To specify files or directories to compare, you can pass command-line arguments to each of these shells. The arguments can point to file or folder paths in directories or archives (supported formats: `zip`, `docx`, and `xlsx`).

For example, to start the comparison between the two Windows directories `c:\Program Files` and `c:\ant`, use the following command:

```
diffDirs.bat "c:\Program Files" "c:\ant"
```

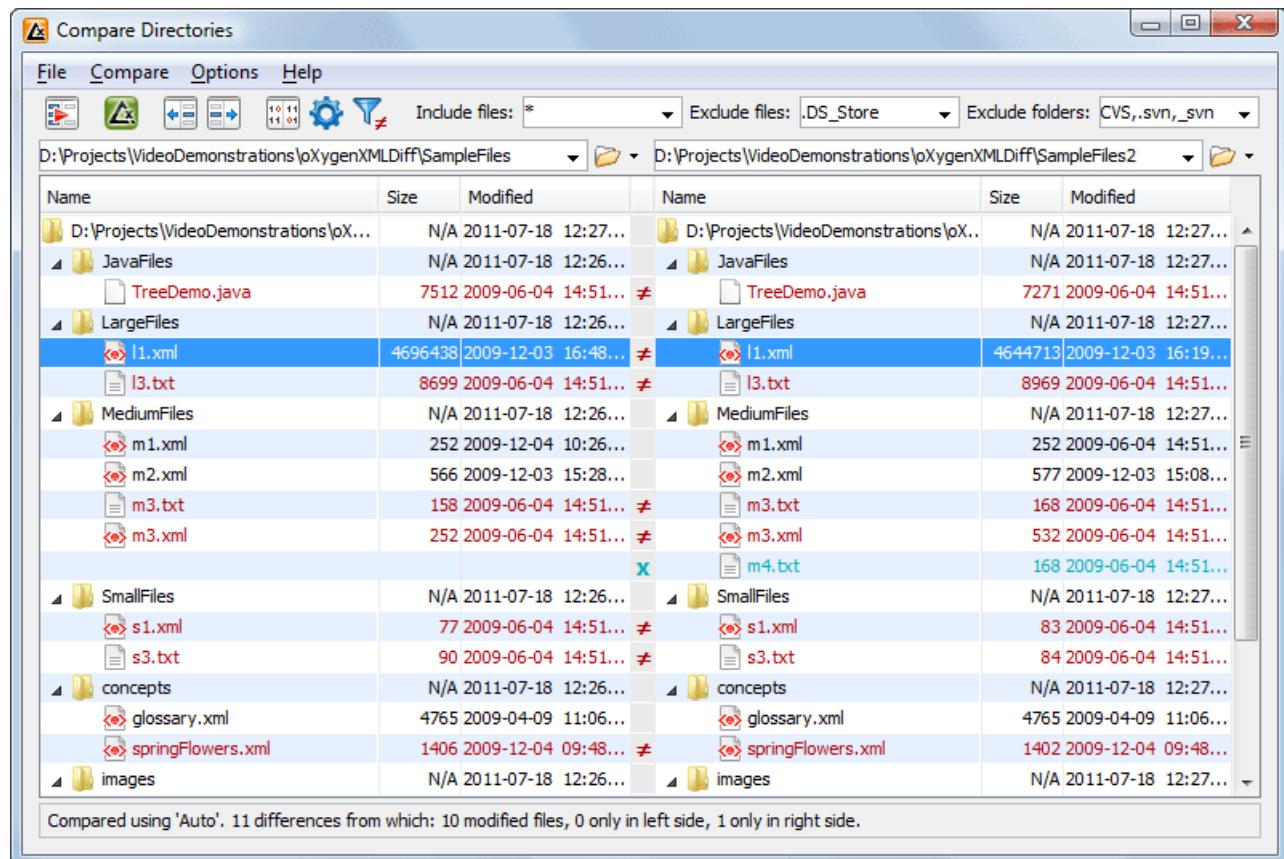
If there are spaces in the path names, surround the paths with quotes. If you pass only one argument, you are prompted to manually choose the second directory or archive. This is also true for the files comparison utility.

## Directories Comparison

The **Compare Directories** tool allows you to compare and manage changes to files and folders within the structure of your directories.

The directories comparison results are presented as a tree of files and directories. The directories and folders that contain files that differ are expanded automatically so that you can focus directly on the differences. You can merge the contents of the directories by using the copy actions. If you double-click (or press **Enter**) on a line with a pair of files, Oxygen XML Editor starts a *file comparison* between the two files, using the **Compare Files** tool.

 **Note:** The comparison is only available for file type associations that are known by Oxygen XML Editor.



**Figure 400: The Compare Directories Window**

### Directories Comparison User Interface

This section explains the user interface of the **Directories Comparison** window.

#### Compare Menu

This menu contains the following actions:

-  **Perform Directories Differencing** - Looks for differences between the two directories displayed in the left and right side of the application window.
-  **Perform Files Differencing** - Compares the currently selected files.
-  **Copy Change from Right to Left** - Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).

- ➡ **Copy Change from Left to Right** - Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).

## Options Menu

- Preferences** - Opens the preferences.
- Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page where you can configure keyboard shortcuts available for menu items.
- Reset Global Options** - Resets options to their default values. Note that this option appears only when the tool is executed as a stand-alone application.
- Import Global Options** - Allows you to import an options set that you have previously exported.
- Export Global Options** - Allows you to export the current options set to a file.

## Help Menu

The **Help** menu contains the following actions:

### Help (**F1**)

Opens the **Help** dialog box.

### Use online help (Enabled by default)

If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Oxygen XML Editor attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

### Report problem...

Opens a dialog box that allows the user to write the description of a problem that was encountered while using the application. You are able to change the URL where the reported problem is sent by using the `com.oxygenxml.report.problems.url` system property. The report is sent in XML format through the `report` parameter of the POST HTTP method.

### Support Center

Opens the Oxygen XML Editor Support Center web page in a browser.

## Compare Toolbar

The toolbar contains the following actions:



**Figure 401: The Compare toolbar**



### Perform directories differencing

Looks for differences between the two directories displayed in the left and right side of the application window.



### Perform files differencing

Compares the currently selected files.



### Copy Change from Right to Left

Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).



### Copy Change from Left to Right

Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).



### **Binary Compare**

Performs a byte-level comparison on the selected files.



### **Diff Options**

Opens the directory comparison preferences page.



### **Show Only Modifications**

Displays a more uncluttered file structure by hiding all identical files.

## **File and folder filters**

Differences can be filtered using three combo boxes: **Include files**, **Exclude files**, and **Exclude folders**. They come with predefined values and are editable to allow custom values. All of them accept multiple comma-separated values and the \* and ? wildcards. For example, to filter out all jpeg and gif image files, edit the **Exclude files** filter box to read \*.jpeg, \*.png. Each filter keeps a list with the latest 15 filters applied in the drop-down list of the filter box.

## **Directories Selector**

To open the directories you want to compare, select a folder from each **Browse for local directory** button. The **Compare Directories** tool keeps track of the folders you are currently working with and those that you opened in this window. You can see and select them from the two drop-down lists.

If you want to compare the content of two archives, you can select the archives from the **Browse for archive file** button.



**Tip:** By default, the supported archives are not treated as directories and the comparison is not performed on the files inside them. To make Oxygen XML Editor treat supported archives as directories, go to the [Diff preferences page](#) and enable the **Look in archives** option.

## **Comparison Results**

The directory comparison results are presented using two tree-like structures showing the files and folders, including their name, size, and modification date.

Name	Size	Modified		Name	Size	Modified
length-bad.xml	140	2010-07-12 16:43:03		length-bad.xml	140	2010-07-12 16:43:03
length-bad1.xml	140	2010-07-12 16:43:04	≠	length-bad1.xml	125	2010-09-02 11:41:14
length-bad2.xml	143	2010-07-12 16:43:04	≠	length-bad2.xml	126	2010-09-02 11:41:07
length-good.xml	141	2010-07-12 16:43:03	≠	length-good.xml	126	2010-09-02 11:41:03
length.dtd	91	2010-07-12 16:43:03	X			
length.sch	648	2010-07-12 16:43:03	X			
name-bad.xml	171	2010-07-12 16:43:03	≠	name-bad.xml	155	2010-09-02 11:40:38
name.dtd	192	2010-07-12 16:43:03	≠	name.dtd	158	2010-09-02 11:42:09
name.sch	566	2010-07-12 16:43:04	≠	name.sch	351	2010-09-02 11:40:50
present-bad.xml	224	2010-07-12 16:43:04	≠	present-bad.xml	171	2010-09-02 11:41:47
present.dtd	130	2010-07-12 16:43:03	≠	present.dtd	60	2010-09-02 11:40:28
present.sch	482	2010-07-12 16:43:03	≠	present.sch	295	2010-09-02 11:41:51
required-bad1.xml	205	2010-07-12 16:43:03	≠	required-bad1.xml	163	2010-09-02 11:41:24
required-bad2.xml	189	2010-07-12 16:43:03	≠	required-bad2.xml	155	2010-09-02 11:41:18
required-good.xml	197	2010-07-12 16:43:03		required-good.xml	197	2010-07-12 16:43:03
required.dtd	128	2010-07-12 16:43:03	≠	required.dtd	91	2010-09-02 11:41:42
required.sch	612	2010-07-12 16:43:03	≠	required.sch	363	2010-09-02 11:41:38
author	N/A	2010-09-01 11:23:14		author	N/A	2010-09-02 11:42:38
author.sch	613	2010-07-12 16:43:04	X			
source1.xml	165	2010-07-12 16:43:04	X			
source2.xml	184	2010-07-12 16:43:04	X			
paragraph	N/A	2010-09-01 11:23:14		paragraph	N/A	2010-09-02 11:39:34

Compared using 'Timestamp (last modified date/time)'. 20 differences from which: 13 modified files, 7 only in left side, 0 only in right side.

**Figure 402: Comparison Results**

A column that contains graphic symbols separates the two tree-like structures. The graphic symbols can be one of the following:

- An X symbol, when a file or a folder exists in only one of the compared directories.
- A ≠ symbol, when a file exists in both directories but the content differs. The same sign appears when a collapsed folder contains differing files.

The color used for the symbol and the directory or file name can be customized in the [Directories Comparison / Appearance](#) preferences page. You can double-click lines marked with the ≠ symbol to open a **Compare Files** window, which shows the differences between the two files.

### Compare Images

If you double-click a line containing two different images, the **Compare images** window is displayed. This dialog box presents the images in the left and right sides, scaled to fit the available view area. You can use the contextual menu actions to scale the images to their original size or scale them down to fit in the view area.

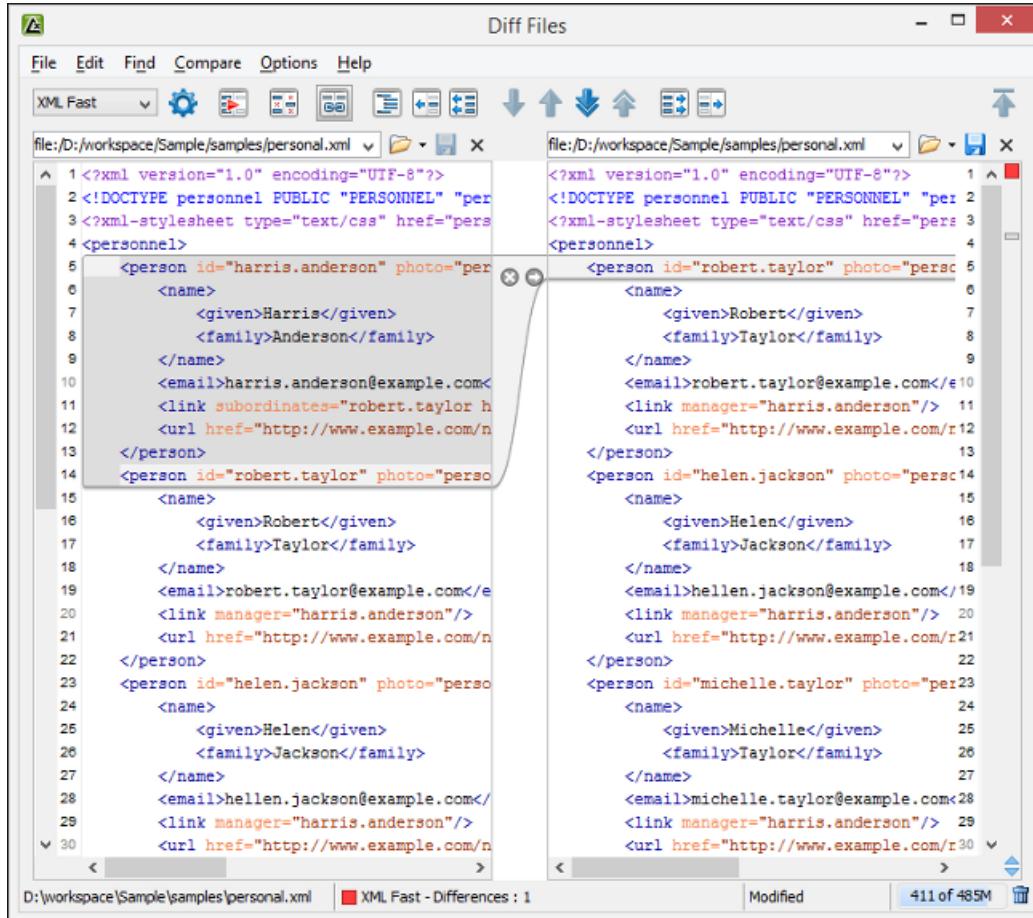
The supported image types are: *GIF, JPG, JPEG, PNG, and BMP*.

### Files Comparison

The **Compare Files** tool can be used to compare files or XML file fragments. To compare two files side by side, open the **Diff Files** dialog box from **Tools > Compare Files**. Using the **Browse** drop-down list, open a file in the left side of the dialog box, and the file you want to compare it to in the right side. To highlight the differences between the two files, click the **Perform File Differencing** button. The line numbers on each side help you to quickly identify the locations of the differences.

To compare XML file fragments, you need to copy and paste the fragments you want to compare into each side, without selecting a file. If a file is already selected, you need to close it, using the **Close (Ctrl W (Command W on OS X))** button, before pasting the fragments.

You can edit both the source and the target file. The differences are refreshed when you save the modified document.



**Figure 403: The Compare Files Window**

Adjacent changes are grouped into blocks of changes. This layout allows you to easily identify and focus on a group of related changes.

When you select a change, a widget containing actions that can be used to copy or append changes from either of the two sides is displayed:

☞ **Append left change to right and ☛ Append right change to left**

Copies the content of the selected change from one side and appends it on the other, after the content of the corresponding change. As a result, the side that the arrows point to will contain the changes from both sides.

⊕ **Copy change from left to right and ⊕ Copy change from right to left**

Replaces the content of a change from one side, with the content of the corresponding change from the other side.

Oxygen XML Editor offers various diff algorithms to compare files or fragments:

- **Auto** - selects the most appropriate algorithm, based on the compared content and its size (selected by default).
- **Characters** - computes the differences at character level.
- **Words** - computes the differences at word level.
- **Lines** - computes the differences at line level.
- **Syntax Aware** - computes differences for file types or XML fragments that are known by Oxygen XML Editor.
- **XML Fast** - works well on large files or fragments at the expense of accuracy.
- **XML Accurate** - works best on small XML files or fragments and offers accurate results, at the expense of speed.

## Main Menu

This section explains the menu actions of the **Files Comparison** window.

### File Menu

The **File** menu of the files comparison tool contain the following actions:

#### Source

The file that is displayed in the left side of the application window.

- **Source >  Open** - Browses for a source file.
- **Source >  Open URL** - Opens the URL to be used as a source file. See [Open URL](#) for details.
- **Source >  Open File from Archive** - Browses an archive content for a source file.
- **Source >  Save** - Saves the changes made in the source file.
- **Source > Save As...** - Displays the **Save As** dialog box that allows you to save the source file with a new name.

#### Target

The file that is displayed in the right side of the application window.

- **Target >  Open** - Browses for a target file.
- **Target >  Open URL** - Opens the URL to be used as a target file. See [Open URL](#) for details.
- **Target >  Open File from Archive** - Browses an archive content for a target file.
- **Target >  Save** - Saves the changes made in the target file.
- **Target > Save As...** - Displays the **Save As** dialog box that allows you to save the target file with a new name.

#### Exit

Quits the application.

### Edit Menu

The following actions are available in the **Edit** menu:



**Cut**  
Cut the selection from the currently focused **Compare** editor to the clipboard.



**Copy**  
Copy the selection from the currently focused **Compare** editor to the clipboard.



**Paste**  
Paste content from the clipboard into the currently focused **Compare** editor.



**Undo**  
Undo changes in the currently focused **Compare** editor.



**Redo**  
Redo changes in the currently focused **Compare** editor.

### Find Menu

The **Find** menu actions are as follows:



**Find/Replace**  
Perform *find/replace* operations in the currently focused **Editor**.



**Find Next**  
Go to the next match using the same options as the last *find* operation. This action runs in both editor panels.

 **Find Previous**

Go to the previous match using the same options as the last *find* operation. This action runs in both editor panels.

**Compare Menu**

The following actions are available in the **Compare** menu:

 **Perform Files Differencing**

Performs a comparison between the source file and target file.

 **Next Block of Changes (Ctrl . (Command . on OS X))**

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes.

 **Note:** A change block groups one or more consecutive lines that contain at least one change.

 **Previous Block of Changes (Ctrl , (Command , on OS X))**

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes.

 **Next Change (Ctrl Shift . (Command Shift . on OS X))**

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change or when there are no changes.

 **Previous Change (Ctrl Shift , (Command Shift , on OS X))**

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change or when there are no changes.

 **Last Change (Ctrl E (Command E on OS X))**

Jumps to the last change.

 **First Change (Ctrl B (Command B on OS X))**

Jumps to the first change.

 **Copy All Changes from Right to Left**

Copies all changes from the target file in the right side to the source file in the left side.

 **Copy All Changes from Left to Right**

Copies all changes from the source file in the left side to the target file in the right side.

 **Copy Change from Right to Left**

Copies the selected difference from the target file in the right side to the source file in the left side.

 **Copy Change from Left to Right**

Copies the selected difference from the source file in the left side to the target file in the right side.

 **Show Word Level Details**

Provides a word-level comparison of the selected change.

 **Show Character Level Details**

Provides a character-level comparison of the selected change.

 **Format and Indent Both Files (Ctrl Shift P (Command Shift P on OS X))**

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

## Options Menu

- **Preferences** - Opens the preferences.
- **Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page where you can configure keyboard shortcuts available for menu items.
- **Reset Global Options** - Resets options to their default values. Note that this option appears only when the tool is executed as a stand-alone application.
- **Import Global Options** - Allows you to import an options set that you have previously exported.
- **Export Global Options** - Allows you to export the current options set to a file.

## Help Menu

The **Help** menu contains the following actions:

### **Help (F1)**

Opens the **Help** dialog box.

### **Use online help (Enabled by default)**

If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Oxygen XML Editor attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

### **Report problem...**

Opens a dialog box that allows the user to write the description of a problem that was encountered while using the application. You are able to change the URL where the reported problem is sent by using the *com.oxygenxml.report.problems.url* system property. The report is sent in XML format through the *report* parameter of the POST HTTP method.

### **Support Center**

Opens the Oxygen XML Editor Support Center web page in a browser.

## Compare Toolbar

This toolbar contains the operations that can be performed on the source and target files or XML fragments.



**Figure 404: The Compare Toolbar**

The following actions are available:

### **Algorithm**

This drop-down list allows you to select one of the following diff algorithms:

- **Auto** - Selects the most appropriate algorithm, based on the compared content and its size (selected by default).
- **Characters** - Computes the differences at character level.
- **Words** - Computes the differences at word level.
- **Lines** - Computes the differences at line level, meaning that it compares two files or fragments looking for identical lines of text. Once identical lines are found, it is considered a match. The content that precedes the match is considered to be a difference and marked accordingly. The algorithm then continues to look for matching lines.
- **Syntax Aware** - Computes differences for known file types or XML fragments. Known file types include those listed in the **New** dialog box, such as XML file types (XSLT files, XSL-FO files, XSD files, RNG files, NVDL files, etc.), XQuery file types (.xquery, .xq, .xqy, .xqm extensions), DTD file types (.dtd, .ent, .mod extensions), TEXT file type (.txt extension), or PHP file type (.php extension).

This algorithm splits the files or fragments into sequences of *tokens* and computes the differences between them. A *token* can have a different meaning, depending on the type of the compared files or fragments . For example:

- when comparing XML files or fragments, a token can be one of the following:
  - the name of an XML tag
  - the '<' character
  - the '>' sequence of characters
  - the name of an attribute inside an XML tag
  - the '=' sign
  - the "" character
  - an attribute value
  - the text string between the start tag and the end tag (a text node that is a child of the XML element corresponding to the XML tag that encloses the text string)
- when comparing plain text, a token can be any continuous sequence of characters or any continuous sequence of whitespaces, including a new line character
- **XML Fast** - Comparison that works well on large files or fragments, but it is less precise than **XML Accurate**.
- **XML Accurate** - Comparison that is more precise than **XML Fast**, at the expense of speed.



### Diff Options

Opens the [Files Comparison page](#).



### Perform directories differencing

Looks for differences between the two directories displayed in the left and right side of the application window.



### Ignore Whitespaces

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.



### Synchronized scrolling

Synchronizes scrolling so that a selected difference can be seen on both sides of the application window. This action enables/disables the previously described behavior.



### Format and Indent Both Files ([Ctrl Shift P \(Command Shift P on OS X\)](#))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.



### Copy Change from Right to Left

Copies the selected difference from the target file in the right side to the source file in the left side.



### Copy All Changes from Right to Left

Copies all changes from the target file in the right side to the source file in the left side.



### Next Block of Changes ([Ctrl . \(Command . on OS X\)](#))

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes.



**Note:** A change block groups one or more consecutive lines that contain at least one change.



### Previous Block of Changes ([Ctrl , \(Command , on OS X\)](#))

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes.

### **Next Change (**Ctrl Shift . (Command Shift . on OS X)**)**

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change or when there are no changes.

### **Previous Change (**Ctrl Shift , (Command Shift , on OS X)**)**

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change or when there are no changes.

### **Copy All Changes from Left to Right**

Copies all changes from the source file in the left side to the target file in the right side.

### **Copy Change from Left to Right**

Copies the selected difference from the source file in the left side to the target file in the right side.

### **First Change (**Ctrl B (Command B on OS X)**)**

Jumps to the first change.

## Files Selector

To open the source and target files to use for comparison, use one of the following options from the  **Browse** drop-down list:

- **Browse for local** - Opens a dialog box to browse for files in your local file system.
- **Browse for remote** - Opens the **Open URL** dialog box to browse for remote files.
- **Browse for archive** - Opens the **Archive Browser** dialog box to browse for archives.
- **Browse Data Source Explorer** - Opens the **Data Source Explorer** dialog box to browse database sources.
- **Search for file** - Opens the **Find Resource** dialog box for advanced search capabilities.

The comparison tool keeps track of the files you are currently working with and those that you opened in this window. You can see and select them from the two combo boxes.

You can also save the changes in either file by clicking the corresponding **Save** button.

You can close compared files by using the **Close** button. To compare XML fragments you need to close the compared files in order to copy and paste the fragments into each side of the panel.

## File Contents Panel

Selected files are opened in two side-by-side editors. A text perspective is used to offer a better view of the differences. You can also compare XML fragments by copying and pasting them into both sides of the panel, without selecting files. To compare XML fragments, if files are already opened you need to close them in order to paste the fragments into the panels.

The two editors are constantly kept in sync. Therefore, if you scroll through the text in one side, the other one also scrolls to show the differences side-by-side. The differences are indicated with highlights connected through colored areas. To navigate through differences, do one of the following:

- Use the navigation buttons on the toolbar.
- Select the navigation options from the **Compare** menu.
- Select a block of differences by clicking its small colored indicator in the overview ruler located in the right-most part of the window. At the top of the overview ruler there is a success indicator that turns green where there are no differences, or red if differences are found.
- Click a colored area in between the two text editors.

You can edit the content in either side of the editor and the differences are refreshed when you save the modified document. If you save modified fragments, rather than a file, a dialog box opens that allows you to save the changes as a new document.

Both editors provide a contextual menu that contains *edit*, *merge*, and *navigation* actions.

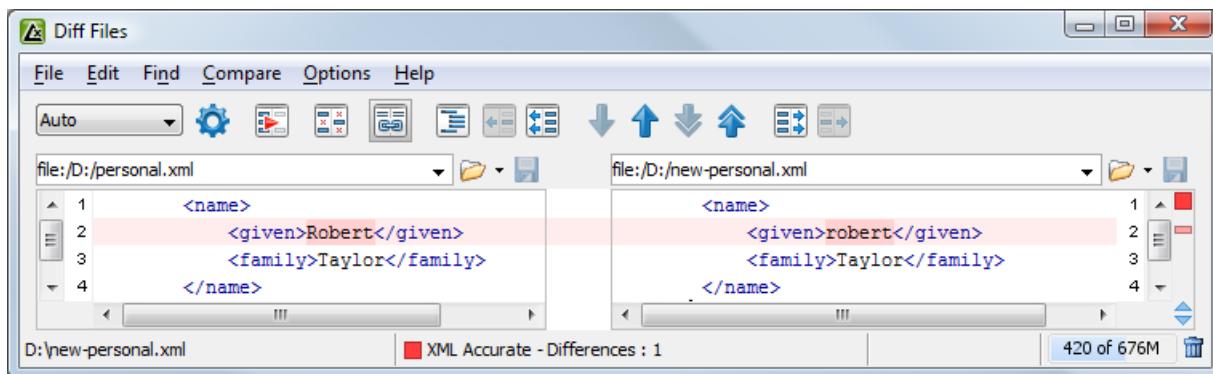
The **Find/Replace** dialog box is displayed by pressing **Ctrl F (Command F on OS X)**. **Find/Replace** options are also available:

- **F3** - Performs a forward search, using the last search configuration.
- **Shift F3** - Performs a backward search, using the last search configuration.

If the compared blocks of text are too large and you want to see the differences at a more fine-tuned level, you can choose options in the **Compare** menu to do a *word level comparison* or *character level comparison*.

### Word Level Comparison

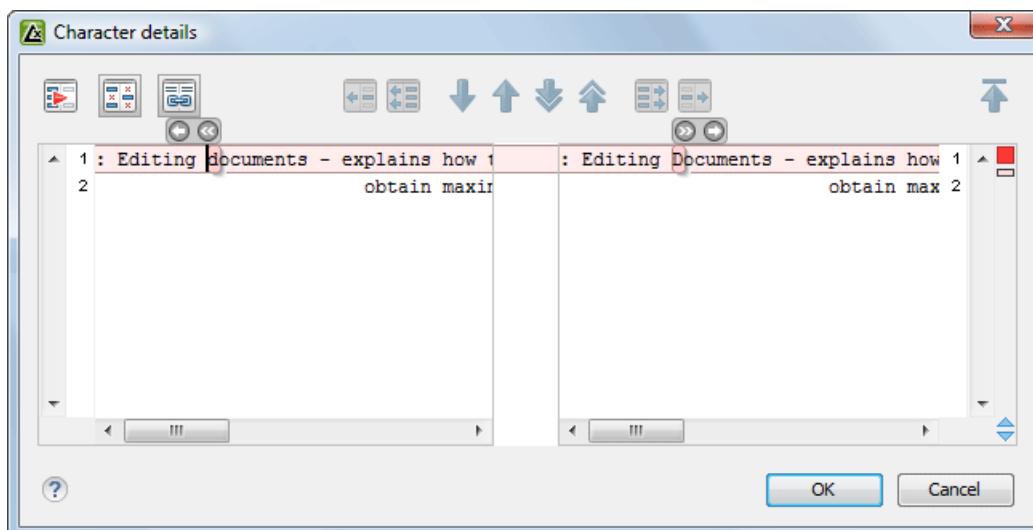
This option is only available if differences exist between the source and the target file. You can do a word level comparison by selecting the **Show word level details** option from the **Compare** menu.



**Figure 405: Word Level Comparison**

### Character Level Comparison

This option is only available if modifications exist between the source and target file. You can do a character level comparison by selecting the **Show Character Level details** option from the **Compare** menu.



**Figure 406: Character Level Comparison**

## XML Digital Signatures

This chapter explains how to apply and verify digital signatures on XML documents.

## Overview

Digital signatures are widely used as security tokens, not just in XML. A digital signature provides a mechanism for assuring integrity of data, the authentication of its signer, and the non-repudiation of the entire signature to an external party:

- A digital signature must provide a way to verify that the data has not been modified or replaced to ensure integrity.
- The signature must provide a way to establish the identity of the data's signer for authentication.
- The signature must provide the ability for the data's integrity and authentication to be provable to a third party for non-repudiation.

A public key system is used to create the digital signature and it's also used for verification. The signature binds the signer to the document because digitally signing a document requires the originator to create a hash of the message and then encrypt that hash value with his own private key. Only the originator has that private key and he is the only one that can encrypt the hash so that it can be unencrypted using his public key. The recipient, upon receiving both the message and the encrypted hash value, can decrypt the hash value, knowing the originator's public key. The recipient must also try to generate the hash value of the message and compare the newly generated hash value with the unencrypted hash value received from the originator. If the hash values are identical, it proves that the originator created the message, because only the actual originator could encrypt the hash value correctly.

XML Signatures can be applied to any digital content (data object), including XML (see W3C Recommendation, [XML-Signature Syntax and Processing](#)). An XML Signature may be applied to the content of one or more resources:

- enveloped or enveloping signatures are applied over data within the same XML document as the signature
- detached signatures are applied over data external to the signature element; the signature is "detached" from the content it signs. This definition typically applies to separate data objects, but it also includes the instance where the signature and data object reside within the same XML document but are sibling elements.

The XML Signature is a method of associating a key with referenced data. It does not normatively specify how keys are associated with persons or institutions, nor the meaning of the data being referenced and signed.

The original data is not actually signed. Instead, the signature is applied to the output of a chain of canonicalization and transformation algorithms, which are applied to the data in a designated sequence. This system provides the flexibility to accommodate whatever "normalization" or desired preprocessing of the data that might be required or desired before subjecting it to being signed.

To canonicalize something means to put it in a standard format that everyone generally uses. Because the signature is dependent on the content it is signing, a signature produced from a not canonicalized document could possibly be different from one produced from a canonicalized document. The canonical form of an XML document is physical representation of the document produced by the method described in this specification. The term canonical XML refers to XML that is in canonical form. The XML canonicalization method is the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term XML canonicalization refers to the process of applying the XML canonicalization method to an XML document or document subset. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would allow the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (and hence the equivalence of the canonical forms over which they were computed) ensures that the information content of the document has not been altered since it was signed.

The following canonicalization algorithms are used in Oxygen XML Editor: Canonical XML (or Inclusive XML Canonicalization)([XMLC14N](#)) and Exclusive XML Canonicalization([EXCC14N](#)). The first is used for XML where the context doesn't change while the second was designed for canonicalization where the context might change.

Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature. In this way all the declarations you might use will be unambiguously specified. A problem appears when the signed XML

is moved into another XML document which has other declarations because the Inclusive Canonicalization will copy them and the signature will be invalid.

Exclusive Canonicalization finds out what namespaces you are actually using (the ones that are a part of the XML syntax) and just copies those. It does not look into attribute values or element content, so the namespace declarations required to process these are not copied.

This type of canonicalization is useful when you have a signed XML document that you wish to insert into other XML documents and it will insure the signature verifies correctly every time, so it is required when you need self-signed structures that support placement within different XML contexts.

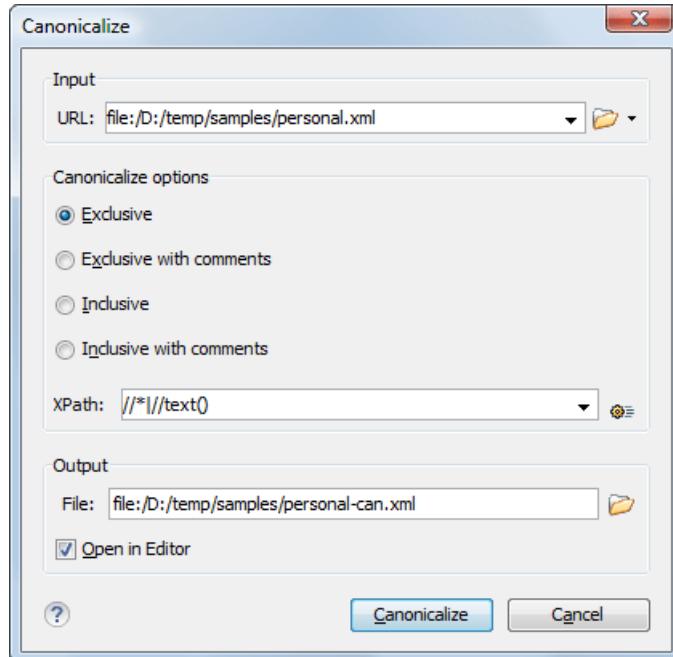
Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it's the safer method from the security perspective because it requires no knowledge of the data that are to be secured in order to safely sign them.

The canonicalization method can specify whether or not comments should be included in the canonical form output by the XML canonicalization method. If a canonical form contains comments corresponding to the comment nodes in the input node-set, the result is called canonical XML with comments. In an uncommented canonical form comments are removed, including delimiter for comments outside document element.

The three operations, **Canonicalize...**, **Sign...**, and **Verify Signature...**, are available from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **Tools** menu.

## Canonicalizing Files

You can select the canonicalization algorithm to be used for a document from the dialog box that is displayed by using the **Canonicalize** action that is available from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **Tools** menu.



**Figure 407: Canonicalization Settings Dialog Box**

You can set the following:

- **URL** - Specifies the location of the input URL.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
- **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.

- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **Output** - Specifies the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

## Certificates

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called *keystores*.

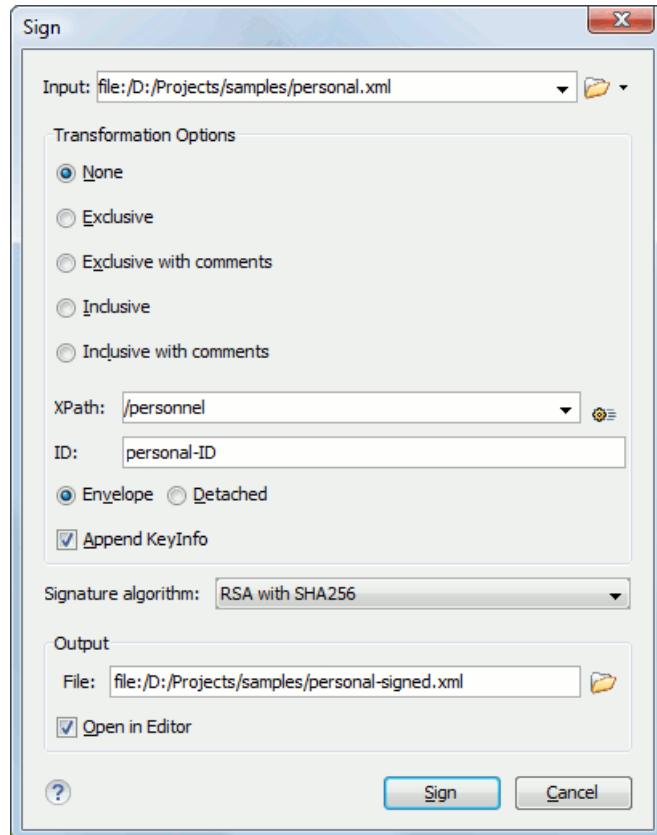
A *keystore* is an encrypted file that contains private keys and certificates. All *keystore* entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No *keystore* can store an entity if its alias already exists in that *keystore* and cannot store trusted certificates generated with keys in its *keystore*.

In Oxygen XML Editor there are provided two types of *keystores*: Java Key Store (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A *keystore* file is protected by a password. In a PKCS 12 *keystore* you should not store a certificate without alias together with other certificates, with or without alias, as in such a case the certificate without alias cannot be extracted from the *keystore*.

To configure the options for a certificate or to validate it, [open the Preferences dialog box](#) and go to **Certificates**.

## Signing Files

You can select the type of signature to be used for documents from a signature settings dialog. To open this dialog, select the **Sign...** action from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **Tools** menu.



**Figure 408: Signature settings dialog**

The following options are available:

- **Input** - Specifies the location of the input URL.
- **Transformation Options:**
  - **None** - If selected, no canonicalization algorithm is used.
  - **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
  - **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
  - **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
  - **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **ID** - Provides ID of the XML element to be signed.
- **Envelope** - If selected, the *enveloped* signature is used.
- **Detached** - If selected, the *detached* signature is used.
- **Append KeyInfo** - If this option is checked, the `ds:KeyInfo` element will be added in the signed document.
- **Signature algorithm** - The algorithm used for signing the document. The following options are available: **RSA with SHA1**, **RSA with SHA256**, **RSA with SHA384**, and **RSA with SHA512**.
- **Output** - Specifies the path of the output file where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.



**Note:** If Oxygen XML Editor could not find a valid certificate, a link is provided at the top of the dialog that opens the [XML Signing Certificates preferences page](#) where you can configure a valid certificate.

Could not obtain a valid certificate. You must configure a valid certificate.

## Verifying the Signature

You can verify the signature of a file by selecting the **Verify Signature** action from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **Tools** menu. The **Verify Signature** dialog then allows you to specify the location of the file whose signature is verified.

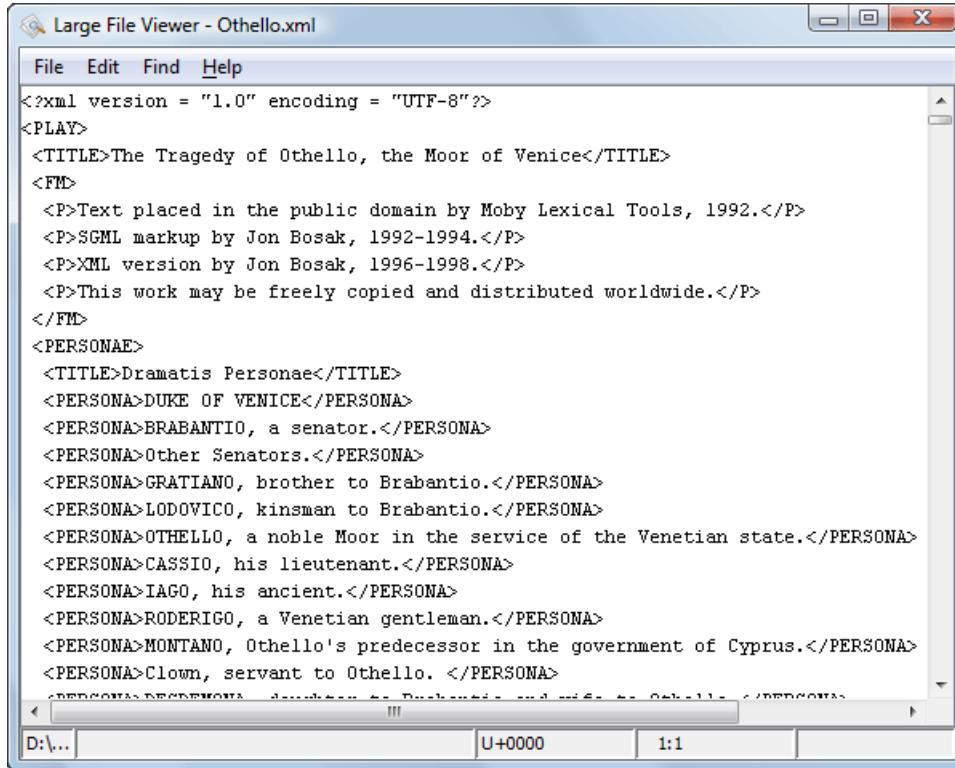
If the signature is valid, a dialog displays the name of the signer. Otherwise, an error shows details about the problem.

## Large File Viewer

XML files tend to become larger and larger mostly because they are frequently used as a format for database export or for porting between different database formats. Traditional XML text editors simply cannot handle opening these huge export files, some having sizes exceeding one gigabyte, because all the file content must be loaded in memory before the user can actually view it.

The best performance of the viewer is obtained for encodings that use a fixed number of bytes per character, like UTF-16 or ASCII. The performance for UTF-8 is very good for documents that use mostly characters of the European languages. For the same encoding, the rendering performance is higher for files consisting of long lines (up to few thousands characters) and may degrade for short lines. In fact, the maximum size of a file that can be rendered in the Large File Viewer decreases when the total number of the text lines of the file increases. Trying to open a very large file, for example a file of 4 GB with a very high number of short lines (100 or 200 characters per line) may produce an *out of memory* error (**OutOfMemoryError**) which would require either increasing the Java heap memory with the `-Xmx` startup parameter or decreasing the total number of lines in the file.

The powerful **Large File Viewer** is available from the **Tools** menu or as a standalone application. You can also right click a file in your project and choose to open it with the viewer. It uses an efficient structure for indexing the opened document. No information from the file is stored in the main memory, just a list of indexes in the file. In this way the viewer can open very large files, up to 10 gigabytes. If the opened file is XML, the encoding used to display the text is detected from the XML prolog of the file. For other file types, the encoding is taken from the Oxygen XML Editor options. See [Encoding for non XML files](#).



**Figure 409: The Large File Viewer**

#### Large File Viewer components:

- The menu bar provides menu driven access to all the features and functions that are available in **Large File Viewer**:

##### **File > Open**

Opens files in the viewer (also available in the contextual pop-up menu).

##### **File > Exit**

Closes the viewer.

##### **Edit > Copy**

Copies the selected text to clipboard (also available in the contextual pop-up menu).

##### **Find > Find**

Opens a reduced **Find** dialog box that provides some basic search options, such as:

- Case sensitive** - When checked, operations are case-sensitive.
- Regular Expression** - When checked, allows you to use any regular expression in *Perl-like syntax*.
- Wrap around** - Continues the find operation from the start (end) of the document after reaching the end (start), depending on whether the search is in forward or (backward) direction.

##### **Help > Help**

Provides access to the User Manual.

- The status bar provides information about the current opened file path, the Unicode representation of the character at caret position and the line and column in the opened document where the caret is located.

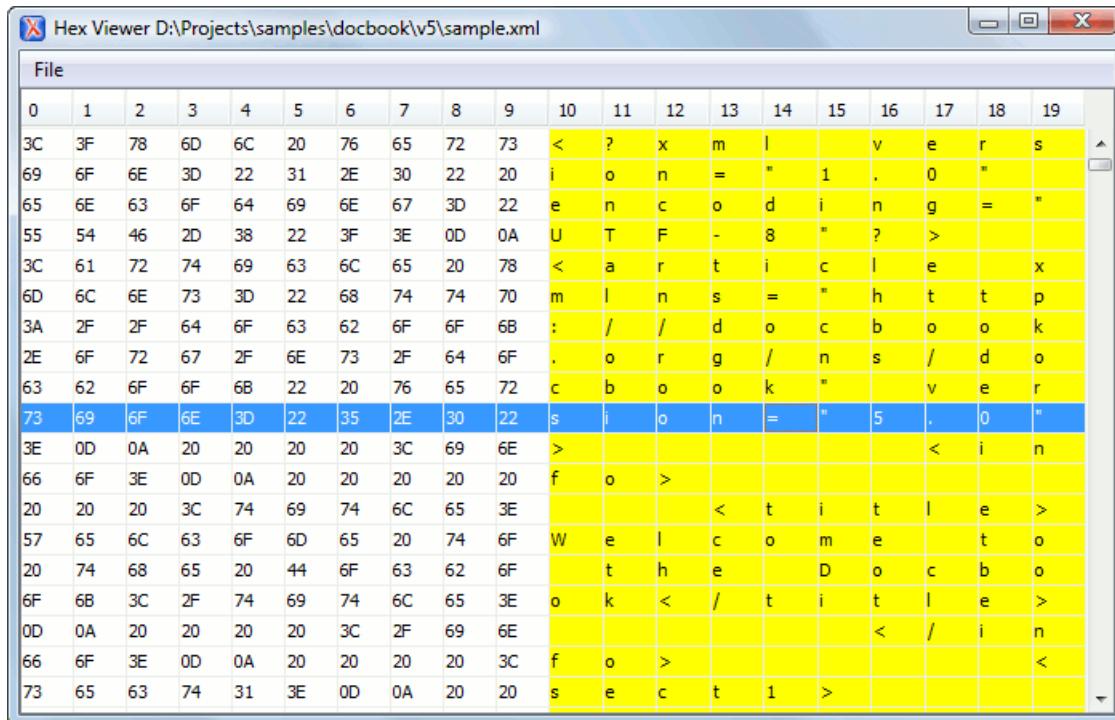
**Attention:** For faster computation the **Large File Viewer** uses a fixed font (plain, monospace font of size 12) to display characters. The font is *not* configurable from the [Preferences page](#).

**Tip:** The best performance of the viewer is accomplished for encodings that use a fixed number of bytes per character, like UTF-16 or ASCII. The performance for UTF-8 is very good for documents that use mostly

characters of the European languages. For the same encoding the rendering performance is high for files consisting of short lines (up to a few thousand characters) and may degrade for long lines.

## Hex Viewer

When the Unicode characters that are visible in a text viewer or editor are not enough and you need to see the byte values of each character of a document, you can start the hex viewer that is available on the **Tools** menu. It has two panels: the characters are rendered in the right panel and the bytes of each character are displayed in the left panel. There is a 1:1 correspondence between the characters and their byte representation: the byte representation of a character is displayed in the same matrix position of the left panel as the character in the matrix of the right panel.



**Figure 410: Hex Viewer**

To open a file in **Hex Viewer** use the **File > Open** action. Alternatively, you can drag a file and drop it in the **Hex Viewer** panel.

## Integrating External Tools

Sometimes an external tool which can be launched from the command line and which is different than a *FO processor* is needed. Oxygen XML Editor offers you the option of integrating such a tool by specifying just the command line for starting the executable file and its working directory. To integrate such a tool, *open the Preferences dialog box* and go to **External Tools**.

If the external tool is applied on one of the files opened in Oxygen XML Editor, *enable the option* for saving all edited files automatically when an external tool is applied.

External tools can be launched from the **External tools** drop-down list in the **Tools** toolbar or from the **Tools > External tools** submenu. While the action is running its icon is a stop icon . When the tool has finished running, it changes the icon back to the original run icon . Please note that even though you can stop the external tool by invoking the action again while it is running, that doesn't mean you can also stop the processes spawned by that external tool. This is

especially a limiting factor when running a batch file as the batch will be stopped but without actually stopping the processes that the batch was running at that time.

### Integrating the Ant Tool

As example let us integrate *the Ant build tool* in Oxygen XML Editor:

- *Download* and *install* Ant on your computer.
- Test your Ant installation from the command-line interface in the directory where you want to use Ant from Oxygen XML Editor, for example run the `clean` target of your `build.xml` file `C:\projects\XMLproject\build.xml`:

```
ant clean
```

- *Open the Preferences dialog box* and go to **External Tools**.
- Create a new external tool entry with the name **Ant tool**, the working directory `C:\projects\XMLproject` and the command line "`C:\projects\XMLproject\ant.bat`" `clean` obtained by browsing to the `ant.bat` file from directory `C:\projects\XMLproject`.
- Run the tool from **Tools > External Tools > Ant tool**. You can see the output in the **Command** results panel:

```
Started: "C:\projects\XMLproject\ant.bat" clean
Buildfile: build.xml

clean:
[echo] Delete output files.
[delete] Deleting 5 files from C:\projects\XMLproject

BUILD SUCCESSFUL
Total time: 1 second
```



---

# Chapter

# 20

---

## Extending Oxygen XML Editor with Plugins

---

**Topics:**

- [\*Introduction\*](#)
- [\*General configuration of an Oxygen XML Editor plugin\*](#)
- [\*Installation\*](#)
- [\*Types of plugin extensions\*](#)
- [\*How to\*](#)
- [\*Example - A Selection Plugin\*](#)
- [\*Creating and Running Automated Tests\*](#)
- [\*Debugging a Plugin Using the Eclipse Workbench\*](#)
- [\*Disabling a Plugin\*](#)

A plugin is a software component which adds extra functionality to the standalone version of the application using a series of application-provided extension points.

This chapter explains how to write and install a plugin for the standalone version of Oxygen XML Editor. The [\*Plugins Development Kit\*](#) contains sample plugins (source and compiled Java code) and the Javadoc API necessary for developing custom plugins.

If you want to customize the Oxygen XML Editor Eclipse Plugin you can look at the [\*Eclipse IDE Integration Sample Project\*](#) to see how an Eclipse plugin can interact with the Oxygen XML Editor APIs.

## Introduction

---

A plugin can have multiple plugin extensions. The following plugin extensions are available (in the order of importance).

Plugins which are used in the entire workspace:

- [Workspace Access Plugin Extension](#) on page 986
- [Components Validation Plugin Extension](#) on page 987
- [Custom Protocol Plugin Extension](#) on page 988
- [Resource Locking Custom Protocol Plugin Extension](#) on page 989
- [Open Redirect Plugin Extension](#) on page 989

Plugins which work only in the Text editing mode:

- [General Plugin Extension](#) on page 991
- [Selection Plugin Extension](#) on page 991
- [Document Plugin Extension](#) on page 992

## General configuration of an Oxygen XML Editor plugin

---

The Oxygen XML Editor functionality can be extended with plugins that implement a clearly specified API.

On the Oxygen XML Editor website there is an [SDK](#) with sample plugins (source and compiled Java code) and the Javadoc API necessary for developing custom plugins.

The minimal implementation of a plugin must provide:

- a Java class that extends the `ro.sync.exml.plugin.Plugin` class
- a Java class that implements the `ro.sync.exml.plugin.PluginExtension` interface
- a plugin descriptor file called `plugin.xml`

A `ro.sync.exml.plugin.PluginDescriptor` object is passed to the constructor of the subclass of the `ro.sync.exml.plugin.Plugin` class. It contains the following data items about the plugin:

- `basedir` - `File` object - the base directory of the plugin.
- `description` - `String` object - the description of the plugin.
- `name` - `String` object - the name of the plugin.
- `vendor` - `String` object - the vendor name of the plugin.
- `version` - `String` object - the plugin version.
- `id` - `String` object - an unique identifier.
- `classLoaderType` - a choice between `preferOxygenResources` (default value) and `preferReferencedResources`. When choosing `preferOxygenResources` the libraries which are referenced in the Oxygen XML Editor `lib` directory will have precedence over those referenced in the `plugin.xml` configuration file, if they have the same package names. When choosing `preferReferencedResources` the libraries which are referenced in the `plugin.xml` configuration file will have precedence over those found in the Oxygen XML Editor `lib` directory, if they have the same package names.

The `ro.sync.exml.plugin.PluginDescriptor` fields are filled with information from the `plugin.xml` configuration file.

The plugin descriptor is an XML file that defines how the plugin is integrated in Oxygen XML Editor and what libraries are loaded. The structure of the plugin descriptor file is fully described in a DTD grammar located in `[OXYGEN_DIR]/plugins/plugin.dtd`. Here is a sample plugin descriptor used by the *Capitalize Lines* sample plugin:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
 name="Capitalize Lines"
```

```

description="Capitalize the first character on each line"
version="1.0.0"
vendor="SyncRO"
class="ro.sync.sample.plugin.caplines.CapLinesPlugin">
<runtime>
 <library name="lib/caplines.jar"/>
</runtime>
<extension type="selectionProcessor"
class="ro.sync.sample.plugin.caplines.CapLinesPluginExtension" keyboardShortcut="ctrl shift EQUALS"/>
</plugin>

```

If your plugin is of type *Selection*, *Document* or *General*, and thus contributes an action either to the contextual menu or to the main menu of the Text editing mode, then you can assign a keyboard shortcut for it. You can use the `keyboardShortcut` attribute for each `extension` element to specify the desired shortcut.

-  **Tip:** To compose string representations of the desired shortcut keys you can go to the Oxygen XML Editor **Menu Shortcut Keys** preferences page, press **Edit** on any action, press the desired key sequence and use the representation that appears in the **Edit** dialog box.

## Referencing libraries

To reference libraries, use either of the following elements:

- `<library name="libraryName" scope="global" />` to point to specific libraries.
  -  **Note:** You can use the  `${oxygenInstallDir}` editor variable as part of the value of the `name` attribute.
- `<librariesFolder name="libraryFolderPath" scope="global" />` to point to multiple libraries located in the specified folder.

Both elements support the `scope` attribute that defines the loading priority. It can have one of the following three values:

- *local* - the library is loaded in the plugin's own class loader. This is the default behaviour
- *global* - the library is loaded in the main application class loader as the last library in the list (as if it would be present in the application `lib` directory)
- *globalHighPriority* - the library is loaded in the main application class loader as the first library in the list (useful to patch certain resources located in other JARs of the application)

## Installation

---

To install a plugin in Oxygen XML Editor, follow these steps:

1. Go to the Oxygen XML Editor installation directory and locate the `plugins` directory.  
The `plugins` directory contains all the plugins available to Oxygen XML Editor.
2. In the `plugins` directory create a subfolder to store the plugin files.
3. In the new folder, place the plugin descriptor file (`plugin.xml`), the Java classes of the plugin and the other files that are referenced in the descriptor file.
4. Restart Oxygen XML Editor.

Alternatively, you can go to **Help > Manage Addons** and use the **Manage Add-ons** dialog box. This dialog allows you to install available plugins and manage the ones that are currently in use.

## Types of plugin extensions

---

A plugin can have one or more defined plugin extensions which provide functionality to the application.

## Workspace Access Plugin Extension

This plugin type allows you to contribute actions to the main menu and toolbars of Oxygen XML Editor, to create custom views and to interact with the application workspace, make modifications to opened documents and add listeners for various events.

Many complex integrations, like integrations with Content Management Systems (CMS) usually requires access to some workspace resources like the toolbar, menus and to the opened XML editors. This type of plugin is also useful because it allows you to make modifications to an opened editor's XML content.

The plugin must implement the interface

`ro.sync.xml.plugin.workspace.WorkspaceAccessPluginExtension`. The callback method

`applicationStarted` of this interface allows access to a parameter of type

`ro.sync.xml.workspace.api.standalone.StandalonePluginWorkspace` which in its turn allows for API access to the application workspace.

The interface `StandalonePluginWorkspace` has three methods which can be called in order to customize the toolbars, menus and views:

- `addToolbarComponentsCustomizer` - Contributes to or modifies existing toolbars. You can specify in the associated `plugin.xml` descriptor additional toolbar IDs using the following construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomWorkspaceAccess">
 <runtime>

 </runtime>

 <extension type="WorkspaceAccess">

 <toolbar id="SampleWorkspaceAccessToolbarID" initialSide="NORTH" initialRow="1"/>
 </extension>
</plugin>
```

The `toolbar` element adds a toolbar in the Oxygen XML Editor interface and allows you to contribute your own plugin specific actions. The following attributes are available:

- `id` - unique identifier of the plugin toolbar
- `initialSide` - specifies the place where the toolbar is initially displayed. The allowed values are `NORTH` and `SOUTH`.
- `initialRow` - specifies the initial row on the specified side where the toolbar is displayed. For example the main menu has an initial row of "0" and the "Edit" toolbar has an initial row of "1".

The `ro.sync.xml.workspace.api.standalone.ToolbarInfo` toolbar component information with the specified id will be provided to you by the customizer interface. You will thus be able to provide Swing components which will appear on the toolbar when the application starts.

- `addViewComponentCustomizer` - Contributes to or modifies existing views or contributes to the reserved custom view. You can specify in the associated `plugin.xml` descriptor additional view IDs using the following construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomWorkspaceAccess">
 <runtime>

 </runtime>

 <extension type="WorkspaceAccess">

 <view id="SampleWorkspaceAccessID" initialSide="WEST" initialRow="0"/>
 </extension>
</plugin>
```

The `view` element adds a view in the Oxygen XML Editor interface and allows you to contribute your own plugin specific UI components. The following attributes are available:

- `id` - unique identifier of the view component
- `initialSide` - specifies the place where the view is initially displayed. The allowed values are `NORTH`, `SOUTH`, `EAST` and `WEST`.

- `initialRow` - specifies the initial row on the specified side where the view is displayed. For example the **Project** view has an initial row of 0 and the Outline view has an initial row of 1. Both views are in the WEST part of the workbench.

The `ro.sync.exml.workspace.api.standalone.ViewInfo` view component information with the specified id will be provided to you by the customizer interface. You will thus be able to provide Swing components which will appear on the view when the application starts.

- `addMenuBarCustomizer` - Contributes to or modifies existing menu components.

Access to the opened editors can be done first by getting access to all URLs opened in the workspace using the API method `StandalonePluginWorkspace.getAllEditorLocations(int editingArea)`. There are two available editing areas: the DITA Maps Manager editing area (available only in Oxygen XML Editor and OxygenXML Author) where only DITA Maps are edited and the main editing area. Using the URL of an opened resource you can gain access to it using the `StandalonePluginWorkspace.getEditorAccess(URL location, int editingArea)` API method. A `ro.sync.exml.workspace.api.editor.WSEditor` allows then access to the current editing page. Special editing API is supported only for the **Text** (`ro.sync.exml.workspace.api.editor.page.text.WSTextEditorPage`) page and the **Author** (`ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPage`) page.

In order to be notified when editors are opened, selected and closed you can use the API method `StandalonePluginWorkspace.addEditorChangeListener` to add a listener.

## Option Page Plugin Extension

This extension type allows developers to add custom preference pages to the application **Preferences** dialog box.

The extension must implement the `ro.sync.exml.plugin.option.OptionPagePluginExtension` interface. The provided callbacks allow the developer to create the custom Swing component which will be added to the page and to react to various calls in order to persistently save the page's settings using the `OptionsStorage` API.

All preferences pages which are contributed by a plugin appear listed in the **Preferences** dialog box in the **Plugins** category. The `plugin.xml` configuration file can specify one or more such extensions using constructs like:

```
<extension type="OptionPage" class="my.package.CustomButtonPagePluginExtension"/>
```

## Components Validation Plugin Extension

This plugin type allows the developer to customize the editor menus, toolbars, and other components by allowing or filtering them from the user interface.

This plugin provides the following API:

- The interface `ComponentsValidatorPluginExtension` - There is one method that must be implemented:
  - `getComponentsValidator()` - Returns a `ro.sync.exml.ComponentsValidator` implementation class used for validating the menus, toolbars, and their actions.
- The `ComponentsValidator` interface provides methods to filter various features from being added to the GUI of Oxygen XML Editor:
  - `validateMenuOrTaggedAction(String[] menuOrActionPath)` - Checks if a menu or a tag action from a menu is allowed and returns a boolean value. A tag is used to uniquely identifying an action. The `String[]` argument is the tag of the menu / action and the tags of its parent menus if any.
  - `validateToolbarTaggedAction(String[] toolbarOrAction)` - Checks if an action from a toolbar is allowed and returns a boolean value. The `String[]` argument is the tag of the action from a toolbar and the tag of its parent toolbar if any.
  - `validateComponent(String key)` - Checks if the given component is allowed and returns a boolean value. The `String` argument is the tag identifying the component. You can remove toolbars entirely using this callback.
  - `validateAccelAction(String category, String tag)` - Checks if the given accelerator action is allowed to appear in the GUI and returns a boolean value. An accelerator action can be uniquely identified

so it will be removed both from toolbars or menus. The first argument represents the action category, the second is the tag of the action.

- `validateContentType(String contentType)` - Checks if the given content type is allowed and returns a boolean value. The String argument represents the content type. You can instruct Oxygen XML Editor to ignore content types like `text / xsl` or `text / xquery`.
- `validateOptionPane(String optionPaneKey)` - Checks if the given options page can be added in the preferences option tree and returns a boolean value. The String argument is the option pane key.
- `validateOption(String optionKey)` - Checks if the given option can be added in the option page and returns a boolean value. The String argument is the option key. This method is mostly used for internal use and it is not called for each option in a preferences page.
- `validateLibrary(String library)` - Checks if the given library is allowed to appear listed in the **About** dialog box and returns a boolean value. The String argument is the library. This method is mostly for internal use.
- `validateNewEditorTemplate(EditorTemplate editorTemplate)` - Checks if the given template for a new editor is allowed and returns a boolean value. The EditorTemplate argument is the editor template. An EditorTemplate is used to create an editor for a given extension. You can thus filter what appears in the list of the **New** dialog box.
- `isDebuggerperspectiveAllowed()` - Check if the debugger perspective is allowed and returns a boolean value.
- `validateSHMarker(String marker)` - Checks if the given marker is allowed and returns a boolean value. The String argument represents the syntax highlight marker to be checked. If you decide to filter certain content types, you can also filter the syntax highlight options so that the content type is no longer present in the Preferences options tree.
- `validateToolbarComposite(String toolbarCompositeTag)` - Checks if the toolbar composite is available. A toolbar composite is a toolbar component such as a drop-down list.



**Tip:** The best way to decide what to filter is to observe the values that Oxygen XML Editor passes when these callbacks are called. You have to create an implementation for this interface which lists in the console all values received by each function. Then you can decide on the values to filter and act accordingly.

## Custom Protocol Plugin Extension

This type of plugins allows the developer to work with a custom designed protocol for retrieving and storing files.

It provides the following API:

- The interface `URLStreamHandlerPluginExtension` - There is one method that must be implemented:
  - `getURLStreamHandler(String protocol)` - It takes as an argument the name of the protocol and returns a `URLStreamHandler` object, or null if there is no URL handler for the specified protocol.
- With the help of the `URLChooserPluginExtension2` interface, it is possible to create your own dialog box that works with the custom protocol. This interface provides two methods:
  - `chooseURLs(StandalonePluginWorkspace workspaceAccess)` - Returns a `URL[ ]` object that contains the URLs the user decided to open with the custom protocol. You can invoke your own URL chooser dialog box here and then return the chosen URLs having your own custom protocol. You have access to the workspace of Oxygen XML Editor.
  - `getMenuItemName()` - Returns a `String` object that is the name of the entry added in the **File** menu.
- With the help of the `URLChooserToolbarExtension` interface, it is possible to provide a toolbar entry which is used for launching the custom URLs chooser from the `URLChooserPluginExtension` implementation. This interface provides two methods:
  - `getToolbarIcon()` - Returns the `javax.swing.Icon` image used on the toolbar.
  - `getToolbarTooltip()` - Returns a `String` that is the tooltip used on the toolbar button.

## Resource Locking Custom Protocol Plugin Extension

This plugin type allows the developer to work with a custom designed protocol for retrieving and storing files. It can lock a resource on opening it in Oxygen XML Editor. This type of plugin extends the custom protocol plugin type with resource locking support.

Such a plugin provides the following API:

- The interface `URLStreamHandlerWithLockPluginExtension` - The plugin receives callbacks following the simple protocol for resource locking and unlocking imposed by Oxygen XML Editor.

There are two additional methods that must be implemented:

- `getLockHandler()` - Returns a `LockHandler` implementation class with the implementation of the lock specific methods from the plugin.
- `isLockingSupported(String protocol)` - Returns a boolean that is `true` if the plugin accepts to manage locking for a certain URL protocol scheme like `ftp`, `http`, `https`, or `customName`.

## XML Refactoring Operations Plugin Extension

This plugin type allows the developer to specify one or more directories from which the XML Refactoring operation resources are loaded.

The `RefactoringOperationsProvider` extension can be used to specify the location where custom XML Refactoring operation resources (XQuery Update scripts and Operation Descriptor files) are stored. Oxygen XML Editor will scan the specified locations to load the custom operations when the XML Refactoring tool is opened, and allows you to share your custom refactoring operations.

### Sample XML Refactoring Operations Plugin Extension

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin PUBLIC "-//Oxygen Plugin" "../plugin.dtd">

<plugin
 id="refactoring.operations"
 name="Refactoring operations plugin"
 description="Contains operation descriptors and related scripts"
 version="1.0">
 <extension type="RefactoringOperationsProvider">
 <folder path="customDir"/>
 <folder path="customDir2"/>
 </extension>
</plugin>
```

## Open Redirect Plugin Extension

This type of plugin is useful for opening more than one file with only one open action.

For example when a zip archive or an ODF file or an OOXML file is open in the **Archive Browser** view a plugin of this type can decide to open a file also from the archive in an XML editor panel. This file can be the `document.xml` main file from an OOXML file archive or a specific XML file from a zip archive.

The plugin must implement the interface `OpenRedirectExtension`. It has only one callback: `redirect(URL)` that receives the URL of the file opened by the Oxygen XML Editor user. If the plugin decides to open also other files it must return an array of information objects (`OpenRedirectInformation[]`) that correspond to these files. Such an information object must contain the URL that is opened in a new editor panel and the content type, for example `text/xml`. The content type is used for determining the type of editor panel. A null content type allows auto-detection of the file type.

## Targeted URL Stream Handler Plugin Extension

This type of plugin can be used when it is necessary to impose custom URL stream handlers for specific URLs.

This plugin extension can handle the following protocols: `http`, `https`, `ftp` or `sftp`, for which Oxygen XML Editor usually provides specific fixed URL stream handlers. If it is set to handle connections for a specific protocol, this extension will be asked to provide the URL stream handler for each opened connection of an URL having that protocol.

To use this type of plugin, you have to implement the `ro.sync.exml.plugin.urlstreamhandler.TargetedURLStreamHandlerPluginExtension` interface, that provides the following methods:

- `boolean canHandleProtocol(String protocol)`

This method checks if the plugin can handle a specific protocol. If this method returns `true` for a specific protocol, the `getURLStreamHandler(URL)` method will be called for each opened connection of an URL having this protocol.

- `URLStreamHandler getURLStreamHandler(URL url)`

This method provides the URL handler for the specified URL and it is called for each opened connection of an URL with a protocol for which the `canHandleProtocol(String)` method returns `true`.

If this method returns `null`, the default Oxygen XML Editor `URLStreamHandler` is used.

To use this type of extension in your plugin, create an extension of `TargetedURLHandler` type in your `plugin.xml` and specify the class that implements `TargetedURLStreamHandlerPluginExtension`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomTargetedURLStreamHandlerPlugin">
 <runtime>

 </runtime>

 <extension type="TargetedURLHandler" class="CustomTargetedURLStreamHandlerPluginExtension"/>

</plugin>
```

This extension can be useful in situations when connections opened from a specific host must be handled in a particular way. For example, the Oxygen XML Editor `HTTP URLStreamHandler` may not be compatible for sending and receiving SOAP using the SUN Web Services implementation. In this case you can override the stream handler set by Oxygen XML Editor for `HTTP` to use the default SUN `URLStreamHandler` which is more compatible with sending and receiving SOAP requests.

```
public class CustomTargetedURLStreamHandlerPluginExtension
 implements TargetedURLStreamHandlerPluginExtension {

 @Override
 public boolean canHandleProtocol(String protocol) {
 boolean handleProtocol = false;
 if ("http".equals(protocol) || "https".equals(protocol)) {
 // This extension handles both HTTP and HTTPS protocols
 handleProtocol = true;
 }
 return handleProtocol;
 }

 @Override
 public URLStreamHandler getURLStreamHandler(URL url) {
 // This method is called only for the URLs with a protocol
 // for which the canHandleProtocol(String) method returns true (HTTP and HTTPS)

 URLStreamHandler handler = null;

 String host = url.getHost();
 String protocol = url.getProtocol();
 if ("some_host".equals(host)) {
 // When there are connections opened from some_host, the SUN HTTP(S)
 // handlers are used
 if ("http".equals(protocol)) {
 handler = new sun.net.www.protocol.http.Handler();
 } else {
 handler = new sun.net.www.protocol.https.Handler();
 }
 }
 return handler;
 }
}
```

## Lock Handler Factory Plugin Extension

This type of extension is used for locking resources from a specific protocol.

It provides the following API:

- The interface `LockHandlerFactoryPluginExtension`.

You need to implement the following two methods:

- `LockHandler getLockHandler()`

Gets the lock handler for the current handled protocol. Might be `null` if not supported.

- `boolean isLockingSupported(String protocol)`

Checks if a lock handler can be provided for a specific protocol.

To use this type of extension in your plugin, create an extension of `LockHandlerFactory` type in your `plugin.xml` and specify the class implementing `LockHandlerFactoryPluginExtension`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomLockHandler">
 <runtime>

 </runtime>
 <extension type="LockHandlerFactory" class="LockHandlerFactoryPluginExtensionImpl"/>

</plugin>
```

## StylesFilter Plugin Extension

This plugin type allows the developer to dynamically modify the CSS styles used to render elements in the **Author** mode.

The plugin must extend the

`ro.sync.exml.plugin.author.css.filter.GeneralStylesFilterExtension` class. This class has a callback on which you can alter the styles for an Author element.

This extension point is similar with the Styles Filter that you set at the *Document Type* level. The only difference is that the plugin filters styles from any opened XML document, regardless of the document type. The changes made by this plugin are prioritised over the changes made by the Document Type level filter.

## Plugin Extensions designed to work only in the Text Editing Mode

These plugin extensions operate only when editing documents in the Text mode. They are mounted automatically by the application on the contextual menu in the **Plugins** submenu.

### General Plugin Extension

This plugin type allows the developer to invoke custom code and to interact with the workspace of Oxygen XML Editor.

This plugin is the most general plugin type. It provides a limited API:

- The interface `GeneralPluginExtension` - Intended for general-purpose plugins, kind of external tools but triggered from the **Plugins** menu. The implementing classes must provide the method `process(GeneralPluginContext)` which must provide the plugin processing. This method takes as a parameter a `GeneralPluginContext` object.
- The class `GeneralPluginContext` - Represents the context in which the general plugin extension does its processing. The `getPluginWorkspace()` method allows you access to the workspace of Oxygen XML Editor.

### Selection Plugin Extension

A selection plugin can be applied to both an XML document and a non-XML document. It works as follows: the user makes a selection in the editor, displays the contextual menu, and selects from the **Plugins** submenu the item corresponding to the plugin.

This plugin type provides the following API:

- The interface `SelectionPluginExtension` - The context containing the selected text is passed to the extension and the processed result is going to replace the initial selection. The `process(GeneralPluginContext)` method must return a `SelectionPluginResult` object which contains the result of the processing. The `String` value returned by the `SelectionPluginResult` object can include editor variables like `$/caret` and `$/selection`.
- The `SelectionPluginContext` object represents the context. It provides four methods:
  - `getSelection()` - Returns a `String` that is the current selection of text.
  - `getFrame()` - Returns a `Frame` that is the editing frame.
  - `getPluginWorkspace()` - Returns access to the workspace of Oxygen XML Editor.
  - `getDocumentURL()` - Returns the URL of the current edited document.

## Document Plugin Extension

This plugin type can be applied only to an XML document. It can modify the current document which is received as callback parameter.

The plugin is started by selecting the corresponding menu item from the contextual menu of the XML editor (Text mode), **Plugins** submenu. It provides the following API:

- The interface `DocumentPluginExtension` - Receives the context object containing the current document in order to be processed. The `process(GeneralPluginContext)` method can return a `DocumentPluginResult` object containing a new document.
- The `DocumentPluginContext` object represents the context. It provides three methods:
  - `getDocument()` - Returns a `javax.swing.text.Document` object that represents the current document.
  - `getFrame()` - Returns a `java.awt.Frame` object that represents the editing frame.
  - `getPluginWorkspace()` - Returns access to the workspace of Oxygen XML Editor.

---

## How to

Different tutorials about how to implement complex plugins.

### How to Write a CMS Integration Plugin

In order to have a complete integration between Oxygen XML Editor and any CMS you usually have to write a plugin which combines two available plugin extensions:

- [Workspace Access](#)
- [Custom protocol](#)

The usual set of requirements for an integration between Oxygen XML Editor and the CMS are the following:

- Contribute to the Oxygen XML Editor toolbars and main menu with your custom **Check Out** and **Check In** actions:
  - **Check Out** triggers your custom dialog boxes that allow you to browse the remote CMS and choose the resources you want to open.
  - **Check In** allows you to send the modified content back to the server.

You can use the **Workspace Access** plugin extension (and provided sample Java code) for all these operations.

- When **Check Out** is called, use the Oxygen XML Editor API to open your custom URLs (URLs created using your custom protocol). It is important to implement and use a **Custom Protocol** extension in order to be notified when the files are opened and saved and to be able to provide the content for the relative references the files may contain to Oxygen XML Editor. Your custom `java.net.URLStreamHandler` implementation checks out the resource content from the server, stores it locally and provides its content. Sample **Check Out** implementation:

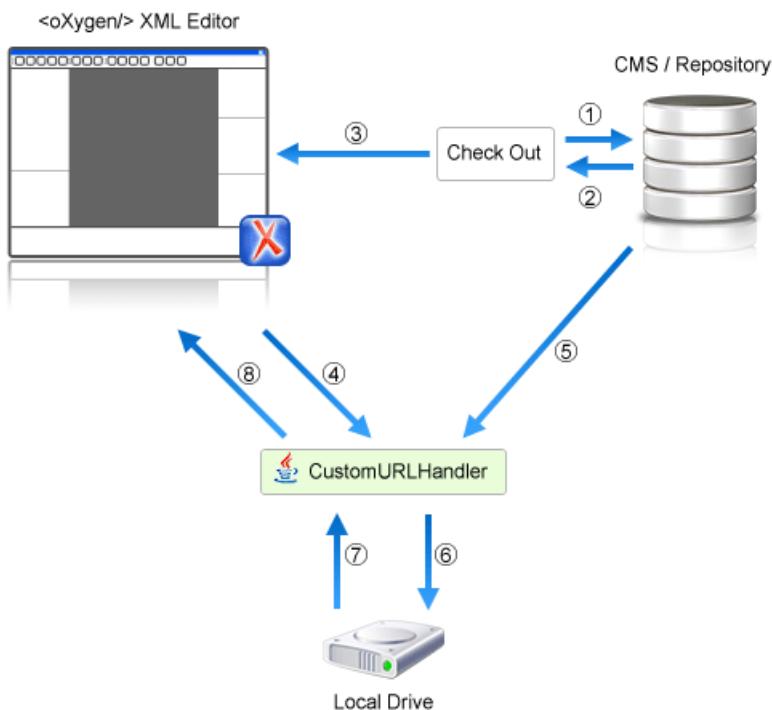
```
/**
 * Sample implementation for the "Check Out" method.
 */
```

```

/*
 * @param pluginWorkspaceAccess The plugin workspace access (Workspace Access plugin).
 * @throws MalformedURLException
 */
private void checkOut(StandalonePluginWorkspace pluginWorkspaceAccess) throws MalformedURLException {
 //TODO Show the user a custom dialog for browsing the CMS
 //TODO after the user selected the resource create an URL with a custom protocol
 // which will uniquely map to the resource on the CMS using the URLHandler
 //something like:
 URL customURL = new URL("mycms://host/path/to/file.xml");
 //Ask Oxygen to open the URL
 pluginWorkspaceAccess.open(customURL);
 //Oxygen will then your custom protocol handler to provide the contents for the resource
 "mycms://host/path/to/file.xml"
 //Your custom protocol handler will check out the file in a temporary directory for example and provide
 the content from it.
 //Oxygen will also pass through your URLHandler if you have any relative references which need to be
 opened/obtained.
}

```

Here is a diagram of the **Check Out** process:



Each phase is described below:

1. Browse CMS repository
  2. User chooses a resource
  3. Use API to open custom URL: mycms://path/to/file.xml
  4. Get content of URL: mycms://path/to/file.xml
  5. Get content of resource
  6. Store on disk for faster access
  7. Retrieve content from disk if already checked out
  8. Retrieved content
- Contribute a special **Browse CMS** action to every dialog box in Oxygen XML Editor where a URL can be chosen to perform a special action (such as the **Insert a DITA Content Reference** or **Insert Image Reference** action).
- Sample code:

```

//Add an additional browse action to all dialogs/places where Oxygen allows selecting an URL.
pluginWorkspaceAccess.addInputURLChooserCustomizer(new InputURLChooserCustomizer() {
 public void customizeBrowseActions(List<Action> existingBrowseActions, final InputURLChooser chooser)
{

```

```

//IMPORTANT, you also need to set a custom icon on the action for situations when its text is not used for display.
Action browseCMS = new AbstractAction("CMS") {
 public void actionPerformed(ActionEvent e) {
 URL chosenResource = browseCMSAndChooseResource();
 if (chosenResource != null) {
 try {
 //Set the chosen resource in the dialog's combo box chooser.
 chooser.urlChosen(chosenResource);
 } catch (MalformedURLException e1) {
 //
 }
 }
 }
};

existingBrowseActions.add(browseCMS);
});
}

```

When inserting references to other resources using the actions already implemented in Oxygen XML Editor, the reference to the resource is made by default relative to the absolute location of the edited XML file. You can gain control over the way in which the reference is made relative for a specific protocol like:

```

//Add a custom relative reference resolver for your custom protocol.
//Usually when inserting references from one URL to another Oxygen makes the inserted path relative.
//If your custom protocol needs special relativization techniques then it should set up a custom relative

//references resolver to be notified when resolving needs to be done.
pluginWorkspaceAccess.addRelativeReferencesResolver(
 //Your custom URL protocol for which you already have a custom URLStreamHandlerPluginExtension set
up.
 "mycms",
 //The relative references resolver
 new RelativeReferenceResolver() {
 public String makeRelative(URL baseURL, URL childURL) {
 //Return the referenced path as absolute for example.
 //return childURL.toString();
 //Or return null for the default behavior.
 return null;
 }
 });

```

- Write the `plugin.xml` descriptor. Your plugin combines the two extensions using a single set of libraries. The descriptor would look like:

```

<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
 name="CustomCMSAccess"
 description="Test"
 version="1.0.0"
 vendor="ACME"
 class="custom.cms.CMSAccessPlugin">
 <runtime>
 <library name="lib/cmsaccess.jar"/>
 </runtime>
 <!--Access to add actions to the main menu and toolbars or to add custom views.-->
 <!--See the "ro.sync.sample.plugin.workspace.CustomWorkspaceAccessPluginExtension" Java sample for more details-->
 <extension type="WorkspaceAccess"
 class="custom.cms.CustomWorkspaceAccessPluginExtension"/>
 <!--The custom URL handler which will communicate with the CMS implementation-->
 <!--See the "ro.sync.sample.plugin.workspace.customprotocol.CustomProtocolURLHandlerExtension" Java sample for more details-->
 <extension type="URLHandler"
 class="custom.cms.CustomProtocolURLHandlerExtension"/>
</plugin>

```

- Create a `cmsaccess.jar` JAR archive containing your implementation classes.
- Copy your new plugin directory in the `plugins` subfolder of the Oxygen XML Editor install folder and start Oxygen XML Editor.

## Class Loading Issues

It is possible that the Java libraries you have specified in the plugin libraries list conflict with the ones already loaded by Oxygen XML Editor. In order to instruct the plugin to prefer its libraries over the ones used by Oxygen XML Editor, you can add the following attribute on the `<plugin>` root element: `classLoaderType="preferReferencedResources"` from the `plugin.xml` descriptor.

**A Late Delegation Class Loader** (the main class loader in Oxygen XML Editor) is a `java.net.URLClassLoader` extension which prefers to search classes in its own libraries list and only if a class is not found there to delegate to the parent class loader.

The main Oxygen XML Editor Class Loader uses as libraries all jars specified in the `[OXYGEN_DIR]\lib` directory. Its parent class loader is the default JVM Class loader. For each instantiated plugin a separate class loader is created having as parent the Oxygen XML Editor Class Loader.

The plugin class loader can be either a standard `java.net.URLClassLoader` or a `LateDelegationClassLoader` (depending on the attribute `classLoaderType` in the `plugin.xml`). Its parent class loader is always the Oxygen XML Editor `LateDelegationClassLoader`.

If you experience additional problems like the following:

```
java.lang.LinkageError: ClassCastException: attempting to cast
jar:file:/C:/jdk1.6.0_06/jre/lib/rt.jar!/javax/xml/ws/spi/Provider.classtojar:file:/D:/Program
Files/Oxygen XML Editor
12/plugins/wspcaccess/.../xdocs/lib/jaxws/jaxws-api.jar!/javax/xml/ws/spi/Provider.class
at javax.xml.ws.spi.Provider.provider(Provider.java:94) at
javax.xml.ws.Service.<init>(Service.java:56)
....
```

The cause could be the fact that some classes are instantiated using the context class loader of the current thread. The most straightforward fix is to write your code in a `try/finally` statement:

```
ClassLoader oldClassLoader = Thread.currentThread().getContextClassLoader();
try {
 //This is the implementation of the WorkspaceAccessPluginExtension plugin interface.
 Thread.currentThread().setContextClassLoader(
 CustomWorkspaceAccessPluginExtension.this.getClass().getClassLoader());
 //WRITE YOUR CODE HERE
} finally {
 Thread.currentThread().setContextClassLoader(oldClassLoader);
}
```

## How to Write A Custom Protocol Plugin

For creating a custom protocol plugin, apply the following steps:

1. Write the handler class for your protocol that implements the `java.net.URLStreamHandler` interface.  
Be careful to provide ways to encode and decode the URLs of your files.
2. Write the plugin class by extending `ro.sync.exml.plugin.Plugin`.
3. Write the plugin extension class that implements the `ro.sync.exml.plugin.urlstreamhandler.URLStreamHandlerPluginExtension` interface.

It is necessary that the plugin extension for the custom protocol implements the `URLStreamHandlerPluginExtension` interface. Without it, you cannot use your plugin, because Oxygen XML Editor is not able to find the protocol handler.

You can choose also to implement the `URLChooserPluginExtension` interface. It allows you to write and display your own customized dialog box for selecting resources that are loaded with the custom protocol.

An implementation of the extension `URLHandlerReadOnlyCheckerExtension` allows you to:

- mark a resource as read-only when it is opened
- switch between marking the resource as read-only and read-write while it is edited

It is useful when opening and editing CMS resources.

4. Write the `plugin.xml` descriptor.  
Remember to set the name of the plugin class to the one from the second step and the plugin extension class name with the one you have chosen at step 3.
5. Create a .jar archive with all these files.
6. Install your new plugin in the `plugins` subfolder of the Oxygen XML Editor install folder.

## How to Pack and Deploy an Add-on

### Packing a Plugin or Framework as an Add-on

This procedure is suitable for developers who want a better control over the add-on package or those who want to automate some of the steps:

1. Pack the plugin or framework as a *ZIP* file or a *JAR*. Please note that you should pack the entire root directory not just its contents.
  2. Digitally sign the package. Please note that you can perform this step only if you have created a *JAR* at the previous step. You will need a certificate signed by a trusted authority. To sign the jar you can either use the `jarsigner` command line tool inside Oracle's Java Development Kit. ([`JDK_DIR`]/bin/`jarsigner.exe`) or, if you are working with *Ant*, you can use the `signjar` task (which is just a front for the `jarsigner` command line tool).
-  **Note:** The benefit of having a signed add-on is that the user can verify the integrity of the add-on issuer. If you don't have such a certificate you can generate one yourself using the `keytool` command line tool. Please note that this approach is mostly recommended for tests since anyone can create a self signed certificate.
3. Create a descriptor file. You can use a template that Oxygen XML Editor provides. To use this template, go to **File > New** and select the **Oxygen add-ons update site** template. Once deployed, this descriptor file is referenced as *update site*.

Alternatively, you can use the **Add-ons Packager** plugin by following this procedure:

1. Install the **Add-ons Packager** plugin from <http://www.oxygenxml.com/InstData/Addons/optional/updateSite.xml> as described in the [Installing Add-ons procedure](#).
2. Restart Oxygen XML Editor. If the add-on is correctly installed, the **Add-ons packager** toolbar action is available.
3. Invoke the **Add-ons packager** toolbar action and input the required information in the displayed dialog box.
4. Press **OK** to complete the packaging process.

### Deploying an Add-on

To deploy an add-on, copy the *ZIP/JAR* file and the descriptor file to an HTTP server. The URL to this location serves as the *Update Site URL*.

## How to Share the Classloader Between a Framework and a Plugin

In some cases you may need to extend the functionality of Oxygen XML Editor both through a framework and through a plugin. Normally, a framework and a plugin both run in their own private classloader. If the framework and the plugin use the same JAVA extensions/classes, it is recommended that they share the same classloader. This way, the common classes are loaded by only one classloader and they will both use the same static objects and have the ability to cast objects between one another.

To do this, [open the \*Preferences\* dialog box](#), go to **Document Type Association**, select the document type, go to the **Classpath** tab, and in the **Use parent classloader from plugin with ID** fields introduce the ID of the plugin. This ID is declared in the [configuration file of the plugin](#).

 **Important:** The shared classed must be specified only in the configuration files of the plugin, and not in the configuration file and the document type class path at the same time.

## Example - A Selection Plugin

The following plugin is called `UppercasePlugin` and is an example of [selection plugin](#). It is used in Oxygen XML Editor for capitalizing the characters in the current selection. This example consists of two Java classes and the plugin descriptor:

- UppercasePlugin.java:

```
package ro.sync.sample.plugin.uppercase;

import ro.sync.exml.plugin.Plugin;
import ro.sync.exml.plugin.PluginDescriptor;

public class UppercasePlugin extends Plugin {
 /**
 * Plugin instance.
 */
 private static UppercasePlugin instance = null;

 /**
 * UppercasePlugin constructor.
 *
 * @param descriptor Plugin descriptor object.
 */
 public UppercasePlugin(PluginDescriptor descriptor) {
 super(descriptor);

 if (instance != null) {
 throw new IllegalStateException("Already instantiated !");
 }
 instance = this;
 }

 /**
 * Get the plugin instance.
 *
 * @return the shared plugin instance.
 */
 public static UppercasePlugin getInstance() {
 return instance;
 }
}
```

- UppercasePluginExtension.java:

```
package ro.sync.sample.plugin.uppercase;

import ro.sync.exml.plugin.selection.SelectionPluginContext;
import ro.sync.exml.plugin.selection.SelectionPluginExtension;
import ro.sync.exml.plugin.selection.SelectionPluginResult;
import ro.sync.exml.plugin.selection.SelectionPluginResultImpl;

public class UppercasePluginExtension implements SelectionPluginExtension {
 /**
 * Convert the text to uppercase.
 *
 * @param context Selection context.
 * @return Uppercase plugin result.
 */
 public SelectionPluginResult process(SelectionPluginContext context) {
 return new SelectionPluginResultImpl(
 context.getSelection().toUpperCase());
 }
}
```

- plugin.xml:

```
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
 name="UpperCase"
 description="Convert the selection to uppercase"
 version="1.0.0"
 vendor="SyncRO"
 class="ro.sync.sample.plugin.uppercase.UppercasePlugin">
 <runtime>
 <library name="lib/uppercase.jar"/>
 </runtime>
 <extension type="selectionProcessor"
 class="ro.sync.sample.plugin.uppercase.UppercasePluginExtension"/>
</plugin>
```

## Creating and Running Automated Tests

If you have developed complex custom plugins and/or document types the best way to test your implementation and insure that further changes will not interfere with the current behavior is to make automated tests for your customization.

An Oxygen XML Editor installation standalone (Author or Editor) comes with a main `oxygen.jar` library located in the `[OXYGEN_DIR]`. That JAR library contains a base class for testing developer customizations named `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

Please see below some steps in order to develop JUnit tests for your customizations using the **Eclipse workbench**:

1. Create a new Eclipse Java project and copy to it the entire contents of the [OXYGEN\_DIR].
  2. Add to the **Java Build Path->Libraries** tab all JAR libraries present in the [OXYGEN\_DIR]/lib directory. Make sure that the main JAR library oxygen.jar or oxygenAuthor.jar is the first one in the Java classpath by moving it up in the **Order and Export** tab.
  3. Click **Add Library** and add the JUnit libraries.
  4. Create a new Java class which extends ro.sync.exml.workspace.api.PluginWorkspaceTCBBase.
  5. Pass on to the constructor of the super class the following parameters:
    - File frameworksFolder The file path to the frameworks directory. It can point to a custom frameworks directory where the custom framework resides.
    - File pluginsFolder The file path to the plugins directory. It can point to a custom plugins directory where the custom plugins resides.
    - String licenseKey The license key used to license the test class.
  6. Create test methods which use the API in the base class to open XML files and perform different actions on them. Your test class could look something like:

```

public class MyTestClass extends PluginWorkspaceTCBase {

 /**
 * Constructor.
 */
 public MyTestClass() throws Exception {
 super(new File("frameworks"), new File("plugins"),
 "-----START-LICENSE-KEY-----\n" +
 "\n" +
 "Registration_Name=Developer\n" +
 "\n" +
 "Company=\n" +
 "\n" +
 "Category=Enterprise\n" +
 "\n" +
 "Component=XML-Editor, XSLT-Debugger, Saxon-SA\n" +
 "\n" +
 "Version=14\n" +
 "\n" +
 "Number_of_Licenses=1\n" +
 "\n" +
 "Date=09-04-2012\n" +
 "\n" +
 "Trial=31\n" +
 "\n" +
 "SGN=MCwCFGNoEGJSeiC3XCYIyalvjzHhGhhqAhRNRPdEu8RIWb8icCJO7HqfVP4++A\\|=\\|=\\n" +
 "\n" +
 "-----END-LICENSE-KEY-----");
 }

 /**
 * <p>Description: TC for opening a file and using the bold operation</p>
 * <p>Bug ID: EXM-20417</p>
 *
 * @author radu_coravu
 *
 * @throws Exception
 */
 public void testOpenFileAndBoldEXM_20417() throws Exception {
 WSEditor ed = open(new File("D:/projects/eXml/test/authorExtensions/dita/sampleSmall.xml").toURL());
 //Move caret
 moveCaretRelativeTo("Context", 1, false);

 //Insert
 invokeAuthorExtensionActionForID("bold");
 assertEquals("<?xml version='1.0' encoding='utf-8'?>\n" +

```

```

 "<!DOCTYPE task PUBLIC \"-//OASIS//DTD DITA Task//EN\""
 + "http://docs.oasis-open.org/dita/v1.1/OS/dtd/task.dtd\">\n" +
 "<task id=\"task1\">\n" +
 " <title>Task title</title>\n" +
 " <prolog/>\n" +
 " <taskbody>\n" +
 " <context>\n" +
 " <p>Context for the current task</p>\n" +
 " </context>\n" +
 " <steps>\n" +
 " <step>\n" +
 " <cmd>Task step.</cmd>\n" +
 " </step>\n" +
 " </steps>\n" +
 " </taskbody>\n" +
 "</task>\n" +
 "", getCurrentEditorXMLContent());
}

```

## Debugging a Plugin Using the Eclipse Workbench

To debug problems in the code of the plugin without having to re-bundle the Java classes of the plugin in a JAR library, follow these steps:

1. Download and unpack an *all platforms standalone version* of Oxygen XML Author/Editor/Developer.



**Note:** The extracted folder name depends on which product variant you have downloaded. For the purpose of this procedure the folder will be referred to as [OXYGEN\_DIR].

2. Set up the Oxygen SDK following [this set of instructions](#).
  3. Create an Eclipse Java Project (let's call it `MyPluginProject`) from one of the sample plugins (the Workspace Access plugin for example).
  4. In the `MyPluginProject` folder, create a folder called `myPlugin`. In this new folder copy the `plugin.xml` from the sample plugin. Modify the added `plugin.xml` to add a library reference to the directory where Eclipse copies the compiled output. To find out where this directory is located, invoke the context menu of the project (in the **Project** view), and go to **Build Path > Configure Build Path...**. Then inspect the value of the **Default output folder** text box.

**Example:** If the compiled output folder is `classes`, then you need to add in the `plugin.xml` the following library reference:

```
<library name=". ./classes" />
```

5. Copy the plugin.dtd from the [OXYGEN\_DIR]/plugins folder in the root MyPluginProject folder.
  6. In the MyPluginProject's build path add external JAR references to all the JAR libraries in the [OXYGEN\_DIR]/lib folder. Now your MyPluginProject should compile successfully.
  7. In the Eclipse IDE, create a new *Java Application* configuration for debugging. Set the **Main class** box to ro.sync.exml.Oxygen. Click the **Arguments** tab and add the following code snippet in the **VM arguments** input box, making sure that the path to the plugins directory is the correct one:

```
-Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m
-XX:MaxPermSize=384m -Dcom.oxygenxml.editor.plugins.dir=D:\projects\MyPluginProject
```



**Note:** If you need to configure the plugin for oXygen XML Author or oXygen XML Developer, set the `com.oxygenxml.app.descriptor` to `ro.sync.exml.AuthorFrameDescriptor` or `ro.sync.exml.DeveloperFrameDescriptor`, respectively.

8. Add a break point in the source of one of your Java classes.
  9. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

## Disabling a Plugin

---

To disable a plugin, use one of the following two methods:

- [Open the Preferences dialog box](#), go to **Plugins**, and deselect the plugin that you want to disable.
- Create an empty file called `plugin.disable` next to the plugin configuration file (`plugin.xml`). The plugin will be disabled and will no longer be loaded by the application on startup.



**Note:** This is useful if you want to temporarily stop work on a plugin and use the application without it.

---

# Chapter

# 21

---

## Configuring Oxygen XML Editor

---

**Topics:**

- [\*Preferences\*](#)
- [\*Importing / Exporting Global Options\*](#)
- [\*Project Level Options\*](#)
- [\*Reset Global Options\*](#)
- [\*Customizing Default Options\*](#)
- [\*Scenarios Management\*](#)
- [\*Editor Variables\*](#)
- [\*Configure Toolbars\*](#)
- [\*Custom System Properties\*](#)
- [\*Localizing the User Interface\*](#)
- [\*Setting a Java Virtual Machine Parameter in the Launcher Configuration File / Start-up Script\*](#)

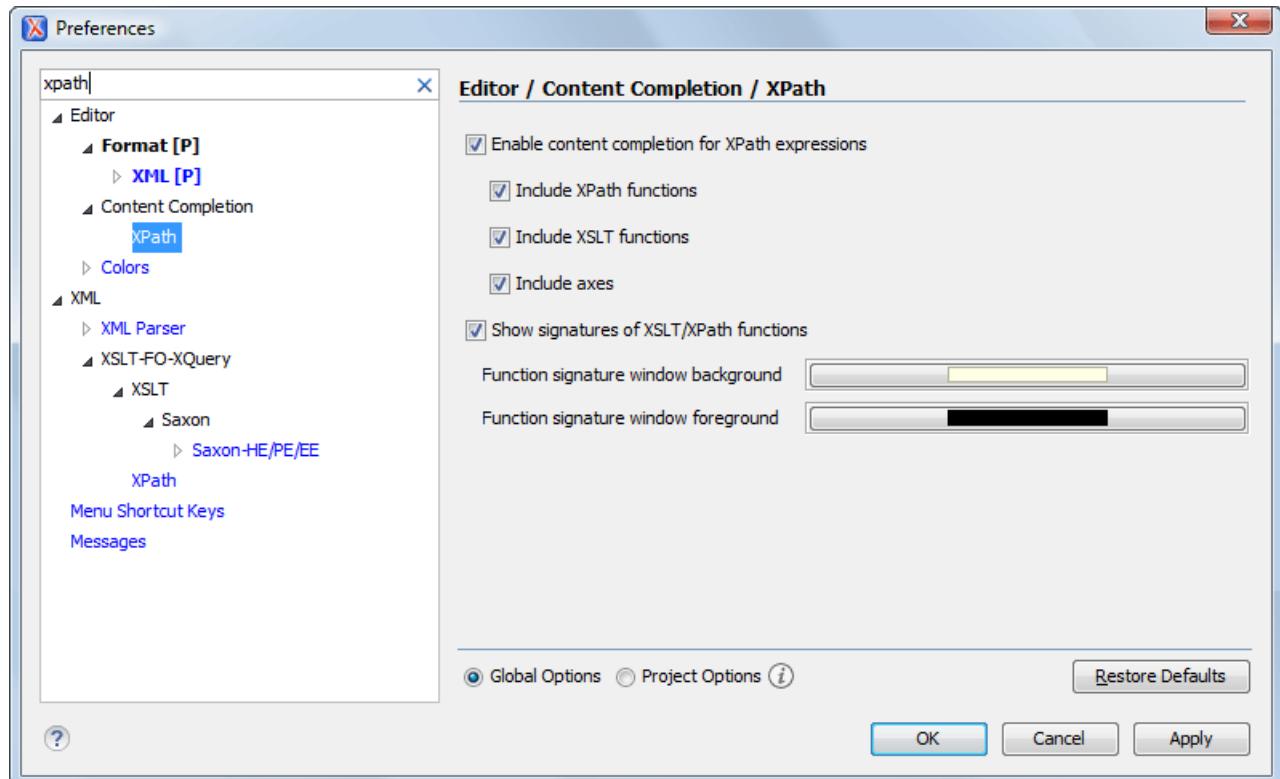
This chapter presents all the user preferences that allow you to configure the application and the editor variables that are available for customizing the user defined commands.

## Preferences

You can configure Oxygen XML Editor options using the **Preferences** dialog box.

To open the preferences dialog box, go to **Options > Preferences**.

You can select the preference page you are interested in from the tree on the left of the **Preferences** dialog box. You can filter the tree by typing in the filter box above the tree.



**Figure 411: The Search Field in the Preferences Dialog Box**

You can restore options to their default values by pressing the **Restore Defaults** button, available in each preferences page.

Press or **F1** for help on any preferences page.

Global options and license information are stored in the following locations:

- [user-home-folder]\Application Data\com.oxygenxml for Windows XP
- [user-home-folder]\AppData\Roaming\com.oxygenxml for Windows Vista/7
- [user-home-folder]/Library/Preferences/com.oxygenxml for Mac OS X
- [user-home-folder]/.com.oxygenxml for Linux

## Global Preferences

The global options cover a number of aspects of the overall operation of . To configure the **Global** options, [open the Preferences dialog box](#) and go to **Global**.

The following options are available in the **Global** preferences page:

- **Automatic Version Checking** - This option sets whether Oxygen XML Editor will check for a new version on startup.
- **Check for notifications** - If enabled, the application will check for various types of messages from the Oxygen XML Editor website and they will be displayed in the status bar. The types of messages include the addition of new videos on the website, the announcement of upcoming webinars and conferences where the Oxygen XML Editor team will participate, and more. This option is enabled by default.
- **Language** - This option sets the language used in the user interface to English, French, German, Dutch, or Japanese. You must restart Oxygen XML Editor for the change to take effect.
- **Other language** - This option sets the language used in the user interface using an interface localization file. For details about creating this file, see [Localizing the User Interface](#) on page 1094. You can use this option to set the language of the user interface to a language that is not shipped with Oxygen XML Editor.



**Note:** If some interface labels are not rendered correctly after restarting the application, (for example Chinese or Korean characters are not displayed correctly), make sure that your operating system has the appropriate language pack installed (for example the East-Asian language pack).

- **Line separator** - This option sets the line separator used when saving files. Use **System Default** to select the normal line separator for your OS. If you want the existing file separator of a file to be maintained, regardless of your current OS, check **Detect the line separator on file open**.
- **Detect the line separator on file open** - When this option is selected, the editor detects the line separator when a file is loaded and it uses it when the file is saved. New files are saved using the line separator defined by the **Line separator** option.
- **Default Internet browser** - This option sets the Web browser that Oxygen XML Editor will use to:
  - open (X)HTML or PDF transformation results
  - open a web page (for example, pointing to specific paragraphs in the W3C recommendation of XML Schema in case of XML Schema validation errors)

If you leave this setting blank, the system default browser will be used.

- **Open last edited files from project** - When this option is enabled, Oxygen XML Editor opens the files you had open the last time you used a project whenever you open the application or switch to that project.
- **Beep on operation finished** - When this option is selected, Oxygen XML Editor beeps when a validation or transform action ends. Different tones are used for success and failure. The tones used may depend on your operating system's sound settings.
- **Show memory status** - When this option is selected, the memory Oxygen XML Editor uses is displayed in the status bar. To free memory, click the **Free unused memory** button located at the right side of the status bar. The memory status bar turns yellow or red when Oxygen XML Editor uses too much memory. You can change the amount of memory available to Oxygen by [changing the parameters of the application launcher](#).
- **Check opened files for file system changes** - When this option is selected, Oxygen XML Editor checks the content of the all opened editors to see if they have been updated by another application. If the file has changed, Oxygen XML Editor will ask you if you want to reload the file.
  - **Auto update unmodified editors on file system changes** - If this option is selected, Oxygen XML Editor automatically updates unmodified editors if the edited file changes externally.
  - **Show Java vendor warning at startup** - If this option is selected, Oxygen XML Editor displays a warning on startup if a non-recommended version of the Java virtual machine is being used.
  - **File Chooser Dialog** - This options sets the directory that will be shown when the [Open file dialog](#) is displayed.
    - **Last visited directory** - The last visited folder will be displayed.
    - **Directory of the edited file** - The folder where the currently edited file is stored will be displayed.
  - **Show hidden files and directories** - If this option is selected, Oxygen XML Editor shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on OS X.

## Apearance Preferences

This preferences page contains a number of options that allow you to change the appearance of the user interface of Oxygen XML Editor. To configure the **Apearance** options, [open the Preferences dialog box](#) and click on **Apearance**.

The following options are available in the **Apearance** preferences page:

- **Look and Feel** - This option allows you to change the graphic style (look and feel) of the user interface. Depending on the operating system, you can choose between different predefined style options.
- **Theme** - This option allows you to set a color theme that will be applied over the entire user interface. You can choose between predefined color themes, and you can also change various options within the selected theme. Links to pages for the various options are displayed in the **Apearance** preferences page. The result of the applied modifications is displayed in the **XML Preview** area.



**Note:** In Windows, if the **Look and Feel** and **Theme** options are set to their default values, Oxygen XML Editor inherits the high contrast theme colors that are set in the operating system.



**Note:** In MAC OS X (starting with Yosemite), if you choose **Graphite** for the **Theme**, it is recommended that you enable the *Use dark menu and Dock* option that is found in **System Preferences > General**.

## Add-ons Preferences

You can use add-ons to enhance the functionality of Oxygen XML Editor. To configure the **Add-ons** options, [open the Preferences dialog box](#) and go to **Add-ons**. The following options are available in the Add-ons preferences page:

- **Enable automatic updates checking** - When this option is selected, Oxygen XML Editor will search for available updates automatically.
- **Add-on Sites URLs** - This is a list of the URLs for the add-on sites. You can add, edit, and delete sites in this list.

## Fonts Preferences

Oxygen XML Editor lets you choose the fonts used in the **Text** and **Grid** editor modes. The fonts for the **Author** mode editor are set in the associated CSS stylesheet. To configure the **Fonts** options, [open the Preferences dialog box](#) and go to **Fonts**.

The following options are available:

- **Editor** - Sets the fonts used in the editor.
- **Note:** On Mac OS X, the default font, Monaco, cannot be rendered in bold.
- **Author default font** - This option sets the default font used **Author** mode. The default font will be overridden by the fonts specified in any CSS file associated with the opened document.
- **Schema default font** - This option sets the font used in:
  - The **Design** mode of the [XML Schema editor](#)
  - Images with schema diagram fragments that are included in the HTML documentation generated from an XML Schema
- **Text antialiasing** - This option sets the text anti-aliasing behavior:
  - **Default** - allows the application to use the setting of the operating system, if available.
  - **On** - sets the text anti-aliasing to pixel level.
  - **Off** - disables text anti-aliasing.
  - sub-pixel anti-aliasing modes, like GASP, LCD\_HRGB, LCD\_HBGR, LCD\_VRGB, and LCD\_VBGR.
- **Text components** - This option sets the font used in text boxes in the interface. After changing the font, restart the application for the change to take full effect.
- **GUI** - This option sets the font used for user interface labels. After changing the font, restart the application for the change to take full effect.

## Document Type Association Preferences

Oxygen XML Editor uses document type associations to associate a [document type](#) with a set of functionality provided by a framework. To configure the **Document Type Association** options, [open the Preferences dialog box](#) and go to **Document Type Association**.

The following actions are available in the preferences panel:

- **Discover more frameworks by using add-ons update sites** - specifies update site URLs for framework add-ons
- **Document types table** - presents the currently defined document type associations, ordered by priority and alphabetically. Each edited document type has a set of association rules (used by the application to detect the proper document type association to use for an opened XML document). A rule is described by:
  - **Namespace** - specifies the namespace of the root element from the association rules set (\* *(any)* by default). If you want to apply the rule only when the root element has no namespace, leave this field empty (remove the ANY\_VALUE string)
  - **Root local name** - specifies the local name of the root element (\* *(any)* by default)
  - **File name** - specifies the name of the file (\* *(any)* by default)
  - **Public ID** - represents the Public ID of the matched document
  - **Java class** - presents the name of the Java class, which is used to determine if a document matches the rule
- **New** - opens a dialog box that allows you to add a new association
- **Edit** - opens a new dialog that allows you to edit an existing association



**Note:** If you try to edit an existing association type when you do not have write permissions to its store location, a dialog box will be shown, asking if you want to extend the document type.

- **Duplicate** - opens a new dialog that allows you to duplicate the configuration of an existing document type association
- **Extend** - extend an existing document type, allowing you to add or remove functionality, starting from a base document type. All of these changes will be saved as a patch. When the base document type is modified and evolves (for example, from one application version to another) the extension will evolve along with the base document type, allowing it to use the new actions added in the base document type.
- **Delete** - deletes the selected associations
- **Enable DTD/XML Schema processing in document type detection** - when this option is enabled, the matching process also examines the DTD/XML Schema associated with the document. For example, the fixed attributes declared in the DTD for the root element are also analyzed, if this is specified in the association rules. This is especially useful if you are writing DITA customizations. DITA topics and maps are also matched by looking for the DITAArchVersion attribute of the root element. This attribute is specified as default in the DTD and it is detected in the root element, helping Oxygen XML Editor to correctly match the DITA customization.

(This option is enabled by default)

- **Only for local DTD's / XML Schemas** - when the previous feature is enabled, you can choose with this option to process only the local DTD's / XML Schemas
- (This option is enabled by default)
- **Enable DTD/XML Schema caching** - when this option is enabled, the associated DTDs or XML Schema are cached when parsed for the first time, improving performance when opening new documents with similar schema associations
- (This option is enabled by default)

## Locations Preferences

Oxygen XML Editor allows you to change the location where [frameworks](#) are stored, and to specify additional framework directories. The **Locations** preferences page allows you to specify the main frameworks folder location. You can choose between the **Default** directory ([ OXYGEN\_DIR ]/frameworks) or a **Custom** specified directory. You can also change the current frameworks folder location value using the com.oxygenxml.editor.frameworks.url system property set either in the application's [.vmoptions configuration files or in the startup scripts](#).

A list of additional frameworks directories can also be specified. The application will look in each of those folders for additional document type configurations to load. Use the **Add**, **Edit** and **Delete** buttons to manage the list of folders.

A document type (configuration) can be loaded from the following locations:

- internal preferences - The document type configuration is stored in the application's [Internal preferences](#)
- additional frameworks directories - The document type configuration is loaded from one of the specified **Additional frameworks directories** list
- add-ons - An add-on can contribute a framework. You can manage the add-ons locations in the [Add-ons preferences page](#)
- the frameworks folder - The main folder containing framework configurations

All loaded document type configurations are first sorted by priority, then by document type name and then by load location (in the exact order specified above). When an XML document is opened, the application chooses the first document type configuration from the sorted list which matches the specific document.

All loaded document type configurations are first sorted by priority, then by document type

### The Document Type Dialog

This dialog allows you to create or edit a *Document Type Association*. The following fields are available in this dialog:

- Name** - the name of the *Document Type Association*
  - Storage** - displays the type of location where the framework configuration file is stored. Can be one of: **External** (framework configuration is saved in a file) or **Internal** (framework configuration is stored in the application's internal options)
-  **Note:** If you set the **Storage** to **Internal** and the document type association settings are already stored in a framework file, the file content is saved in the application's internal options and the file is removed.
- Description** - a detailed description of the framework
  - Priority** - depending on the priority level, Oxygen XML Editor establishes the order in which the existing document type associations are evaluated to determine the type of a document you are opening. It can be one of the following: Lowest, Low, Normal, High, or Highest. You can set a higher priority to *Document Type Associations* you want to be evaluated first.
  - Initial edit mode** - sets the default edit mode when you open a document for the first time

You are able to configure the options of each *framework* in the following tabs:

- [Association rules](#)
- [Schema](#)
- [Classpath](#)
- [Author](#)
- [Templates](#)
- [Catalogs](#)
- [Transformation](#)
- [Validation](#)
- [Extensions](#)

### The Association Rules Tab

By combining multiple association rules you can instruct Oxygen XML Editor to identify the type of a document. An Oxygen XML Editor *association rule* holds information about *Namespace*, *Root local name*, *File name*, *Public ID*, *Attribute*, and *Java class*. Oxygen XML Editor identifies the type of a document when the document matches at least one of the *association rules*. Using the **Document type rule** dialog box, you can create *association rules* that activate on any document matching all the criteria in the dialog box.

In the **Association rules** tab you can perform the following actions:

 **New**

Opens the **Document type rule** dialog box allowing you to create *association rules*.

 **Edit**

Opens the **Document type rule** dialog box allowing you to edit the properties of the currently selected *association rule*.

 **Delete**

Deletes the currently selected *association rules*.

 **Move Up**

Moves the selection to the previous *association rule*.

 **Move Down**

Moves the selection to the following *association rule*.

### The Schema Tab

In the **Schema** tab you can specify a schema that Oxygen XML Editor uses in case an XML document does not contain a schema declaration and no default validation scenario is associated with it.

To set the **Schema URL**, use *editor variables* to specify the path to the Schema file.

 **Note:** It is a good practice to store all resources in the framework directory and use the \${framework} editor variable to reference them. This is a recommended approach to designing a self-contained document type that can be easily maintained and shared between different users.

### The Classpath Tab

The **Classpath** tab displays a list of folders and JAR libraries that hold implementations for API extensions, implementations for custom Author operations, different resources (such as stylesheets), and framework translation files. Oxygen XML Editor loads the resources looking in the folders in the order they appear in the list.

In the **Classpath** tab you can perform the following actions:

 **New**

Opens a dialog box that allows you to add a resource in the **Classpath** tab.

 **Edit**

Opens a dialog box that allows you to edit a resource in the **Classpath** tab.

 **Delete**

Deletes the currently selected resource.

 **Move Up**

Moves the selection to the previous resource.

 **Move Down**

Moves the selection to the following resource.

The **Use parent classloader from plugin with ID** specifies the ID of a plugin. The current framework has access to the classes loaded for the plugin.

### The Author Tab

The **Author** tab is a container that holds information regarding the CSS file used to render a document in the **Author** mode, and regarding framework-specific actions, menus, contextual menus, toolbars, and content completion list of proposals.

The options that you configure in the **Author** tab are grouped in the following sub-tabs:**CSS, Actions, Menu, Contextual menu, Toolbar, Content Completion**.

## CSS

The **CSS** sub-tab contains the CSS files that Oxygen XML Editor uses to render a document in the **Author** mode. In this sub-tab, you can set alternate CSS files. When you are editing a document in the **Author** mode, you can switch between these CSS files from the **Styles** drop-down list on the **Author Styles** toolbar.

The following actions are available in the **CSS** sub-tab:

 **New**

Opens a dialog that allows you to add a CSS file.

 **Edit**

Opens a dialog that allows you to edit a CSS file.

 **Delete**

Deletes the currently selected CSS file.

 **Move Up**

Moves the selection to the previous CSS file.

 **Move Down**

Moves the selection to the following CSS file.

### Enable multiple selection of alternate CSSs

Allows users to apply multiple alternate styles, as layers, over the main CSS style. This option is enabled by default for DITA document types.

### ignore CSSs from the associated document type

The CSS files set in the CSS tab are overwritten by the CSS files specified in the document itself.

### merge them with CSSs from the associated document type

The CSS files set in the CSS tab are merged with the CSS files specified in the document itself.

## Actions

The **Actions** sub-tab holds the framework specific actions. Each action has an unique ID, a name, a description, and a shortcut key.

The following actions are available in this sub-tab:

 **New**

Opens [the Action dialog](#) that allows you to add an action.

 **Duplicate**

Duplicates the currently selected action.

 **Edit**

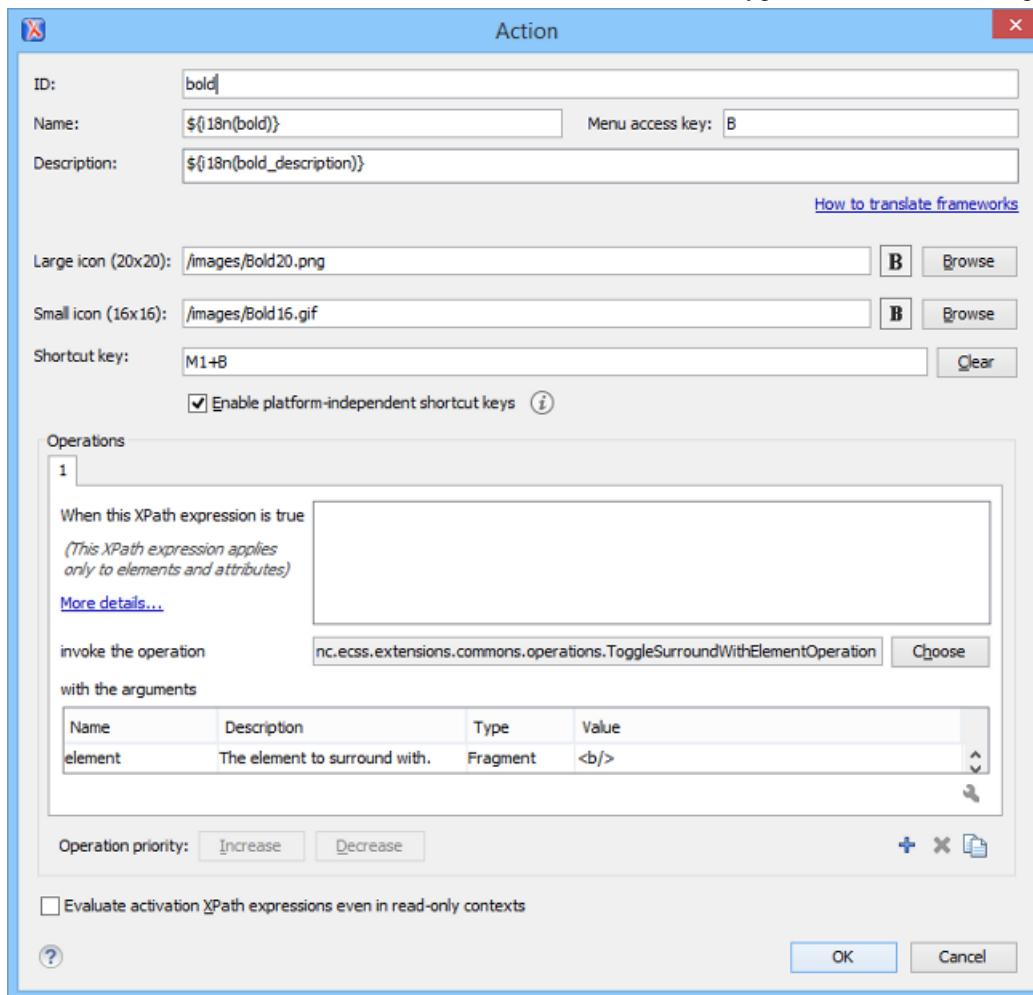
Opens a dialog that allows you to edit an existing action.

 **Delete**

Deletes the currently selected action.

## The Action Dialog Box

To edit an existing document type action or create a new one, [open the Preferences dialog box](#), go to **Document Type Association**, select a document type, and click **Edit** or **New**. The **Document type** dialog box is presented. In this dialog box, go to the **Author** tab, click **Actions**, select an action, and click  **Edit**, or to create a new action click  **New**.



**Figure 412: The Action Dialog Box**

The following options are available in the **Action** dialog box:

- **ID** - Specifies a unique action identifier.
  - **Name** - Specifies the name of the action. This name is displayed as a tooltip or as a menu item.
  - **Menu access key** - In Windows, the menu items are accessed using the **Alt "Letter"** shortcut when the menu is visible. The letter is visually represented by underlining the first occurrence of the letter in the menu item **Name**.
  - **Description** - A description of the action.
  - **Large icon** - Allows you to select an image for the icon that Oxygen XML Editor uses for the toolbar action.
- Tip:** A good practice is to store the image files inside the framework directory and use the \${frameworks} editor variable to make the image relative to the framework location. If the images are bundled in a *jar* archive (for instance, along with some Java operations implementation), it is convenient to reference the images by their relative path location in the *class-path*.
- **Small icon** - Allows you to select an image for the icon that Oxygen XML Editor uses for the contextual menu action.
- Note:** If you are using a Retina or HiDPI display, Oxygen XML Editor automatically searches for higher resolution icons in the path specified in both the **Large icon** and **Small icon** options. For more information, see [the Adding Retina/HiDPI Icons in a Framework section](#).

- **Shortcut key** - This field allows you to configure a shortcut key for the action that you are editing. The + character separates the keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut that you specify in this field is platform-independent and the following modifiers are used:
  - M1 represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
  - M2 represents the **Shift** key.
  - M3 represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
  - M4 represents the **Ctrl** key on MacOS X, and is undefined on other platforms.
- **Operations**

In this section of the **Action** dialog box, you configure the functionality of the action that you are editing. An action has one or more operation modes. The evaluation of an XPath expression activates an operation mode. The first enabled operation mode is activated when you trigger the action. The scope of the XPath expression must consist only of element nodes and attribute nodes of the edited document. Otherwise, the XPath expression does not return a match and does not fire the action. For more details see: [Action Mode Activation using XPath Expressions](#) on page 1010.

The following options are available in this section:

- **When this XPath expression is true** - An XPath 2.0 expression that applies to elements and attributes. For more details see: [Action Mode Activation using XPath Expressions](#) on page 1010.
- **invoke the operation** - Specifies the invoked operation.
- **with the arguments** - Specifies the arguments of the invoked operation.
- **Edit** - Allows you to edit the arguments of the operation.
- **Operation priority** - Increases or decreases the priority of an operation. The operations are invoked in the order of their priority. If more than one XPath expression is true, the operation with the highest priority is invoked.
  - **Add** - Adds an operation.
  - **Remove** - Removes an operation.
  - **Duplicate** -Duplicates an operation.
- **Evaluate activation XPath expressions even in read-only contexts** - If this checkbox is enabled, the action can be invoked even when the caret is placed in a read-only location.

## Action Mode Activation using XPath Expressions

An Author extension action can have multiple modes, each mode invoking an Author Operation with certain configured parameters. Each action mode has an XPath 2.0 expression for activating it.

For each action mode the application will check if the XPath expression is fulfilled (when it returns a not empty nodes set or a *true* result). If it is fulfilled, the operation defined in the action mode will be executed.

Two special XPath extension functions are provided: the `oxy:allows-child-element()` function that you can use to check whether an element is valid in the current context, considering the associated schema and the `oxy:current-selected-element()` function that you can use to get the currently selected element.

### The `oxy:allows-child-element()` Function

This extension function allows author actions to be available in a context only if the associated schema permits it.

The `oxy:allows-child-element()` is evaluated at the caret position and has the following signature:  
`oxy:allows-child-element($childName, ($attributeName, $defaultValue, $contains?))?`.

The following parameters are supported:

#### **childName**

the name of the element that you want to check whether it is valid in the current context. Its value is a string that supports the following forms:

- the child element with the specified local name that belongs to the default namespace.

```
oxy:allows-child-element("para")
```

The above example verifies if the para element (of the default namespace) is allowed in the current context.

- the child element with the local name specified by any namespace.

```
oxy:allows-child-element("*:para")
```

The above example verifies if the para element (of any namespace) is allowed in the current context.

- a qualified name of an element.

```
oxy:allows-child-element("prefix:para")
```

The prefix is resolved in the context of the element where the caret is located. The function matches on the element with the para local name from the previous resolved namespace. In case the prefix is not resolved to a namespace, the function returns false.

- any element.

```
oxy:allows-child-element("*")
```

The above function verifies if any element is allowed in the current context.

 **Note:** A common use case of `oxy:allows-child-element("*)` is in combination with the `attributeName` parameter.

### **attributeName**

the attribute of an element that you want to check whether it is valid in the current context. Its value is a string that supports the following forms:

- the attribute with the specified name from no namespace.

```
oxy:allows-child-element("", "class", "topic/topic")
```

The above example verifies if an element with the class attribute and the default value of this attribute (that contains the topic/topic string) is allowed in the current context.

- the attribute with the local name specified by any namespace.

```
oxy:allows-child-element("", "*:localname", "topic/topic")
```

- a qualified name of an attribute.

```
oxy:allows-child-element("", "prefix:localname", "topic/topic")
```

The prefix is resolved in the context of the element where the caret is located. In case the prefix is not resolved to a namespace, the function returns false.

### **defaultValue**

a string that represents the default value of the attribute. Depending on the value of the next parameter the default value of the attribute must either contain this value or be equal with it.

### **contains**

an optional boolean. The default value is true. For the true value, the default value of the attribute must contain the defaultValue parameter. In case the value is false, the two values must be the same.

The `oxy:current-selected-element()` Function

This function returns the fully selected element. In case no element is selected, the function returns an empty sequence.

```
oxy:current-selected-element()[self::p]/b
```

This example returns the `b` elements that are children of the currently selected `p` element.

## Menu

In the **Menu** sub-tab you configure what *framework* specific actions appear in the Oxygen XML Editor menu. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Editor menu.

To add an action in this section as a sibling of the currently selected action, use the  **Add as sibling** button. To add an image in this section as a child of the currently selected action use the  **Add as child** button.

The following actions are available in the **Current actions** section:

 **Edit**

Edits an item.

 **Remove**

Removes an item.

 **Move Up**

Moves an item up.

 **Move Down**

Moves an item down.

## Contextual menu

In the **Contextual menu** sub-tab you configure what framework-specific action the **Content Completion Assistant** proposes. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the contextual menu of a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the  **Add as sibling** button. To add an action in this section as a child of the currently selected action use the  **Add as child** button.

The following actions are available in the **Current actions** section:

 **Edit**

Edits an item.

 **Remove**

Removes an item.

 **Move Up**

Moves an item up.

 **Move Down**

Moves an item down.

## Toolbar

In the **Toolbar** sub-tab you configure what framework-specific action the Oxygen XML Editor toolbar holds. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Editor toolbar when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the  **Add as sibling** button. To add an action in this section as a child of the currently selected action use the  **Add as child** button.

The following actions are available in the **Current actions** section:

 **Edit**

Edits an item.

 **Remove**

Removes an item.

 **Move Up**

Moves an item up.

 **Move Down**

Moves an item down.

#### *Content Completion*

In the **Content Completion** sub-tab you configure what framework-specific the **Content Completion Assistant** proposes. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that the **Content Completion Assistant** proposes when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the  **Add as sibling** button. To add an action in this section as a child of the currently selected action use the  **Add as child** button.

The following actions are available in the **Current actions** section:

 **Edit**

Edits an item.

 **Remove**

Removes an item.

 **Move Up**

Moves an item up.

 **Move Down**

Moves an item down.

#### **The Templates Tab**

The **Templates** tab specifies a list of directories in which new file templates are located. These file templates are gathered from all the document types and presented in the **New** document dialog box.

#### **The Catalogs Tab**

The **Catalogs** tab specifies a list of [XML catalogs](#) which are added to all the catalogs that Oxygen XML Editor uses to resolve resources.

#### **The Transformation Tab**

In the **Transformation** tab you configure the transformation scenarios associated with the framework you are editing. These are the transformation scenarios that are presented in the **Configure Transformation Scenarios** dialog box as associated with the type of the edited document.

You can set one or more of the scenarios from the **Transformation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog box and are also displayed on the tooltip of the **Apply transformation Scenario(s)**.

The **Transformation** tab offers the following options:

 **New**

Opens the **New scenario** dialog box allowing you to create a new transformation scenario.

 **Edit**

Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected transformation scenario.

 **Delete**

Deletes the currently selected transformation scenario.

 **Import scenarios**

Imports transformation scenarios.

 **Export selected scenarios**

Export transformation scenarios.

 **Move Up**

Moves the selection to the previous scenario.

 **Move Down**

Moves the selection to the next scenario.

### The Validation Tab

In the **Validation** tab you configure the validation scenarios associated with the framework you are editing. These are the validation scenarios that are presented in the **Configure Validation Scenarios** dialog box as associated with the type of the edited document.

You can set one or more of the scenarios from the **Validation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog box and are also displayed on the tooltip of the **Apply transformation Scenario(s)** button.

The **Validation** tab offers the following options:

 **New**

Opens the **New scenario** dialog box allowing you to create a new validation scenario.

 **Edit**

Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected validation scenario.

 **Delete**

Deletes the currently selected validation scenario.

 **Import scenarios**

Imports transformation scenarios.

 **Export selected scenarios**

Export transformation scenarios.

 **Move Up**

Moves the selection to the previous scenario.

 **Move Down**

Moves the selection to the next scenario.

### The Extensions Tab

The **Extension** tab specifies implementations of Java interfaces used to provide advanced functionality to the document type.

Libraries containing the implementations must be present in the *classpath* of your document type. The Javadoc available at <http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/> contains details about how each API implementation functions.

## Application Layout Preferences

Oxygen XML Editor offers various perspectives and views that you can arrange in different layouts to suit your needs.

To configure the application layout options, [open the Preferences dialog box](#) and go to **Application Layout**. The following options are available:

- **Select application layout** section
  - **Default** - Uses the default layout for all perspectives. Any modification of this layout (such as closing views, displaying views, or a new view arrangement) is saved on exit and reloaded at start-up.
  - **Predefined** - Allows you to choose one of the predefined layouts:
    - **Advanced** - All views are shown.
    - **Author** - An authoring-oriented layout, which displays views such as **Project**, **Archive Browser**, **DITA Maps Manager**, **Outline**, **Attributes**, **Model**, and **Elements**.
    - **Basic** - Only the **Project** view and the **Outline** view are visible. Recommended when you edit XML content and you need maximum screen space.
    - **Schema development** - The **Project**, **Component Dependencies**, **Resource Hierarchy/Dependencies**, **Outline**, **Palette**, and **Attributes** views are shown.
    - **XQuery development** - The **Project**, **Outline**, **Transformation Scenarios**, **XSLT/XQuery input** views are shown.
    - **XSLT development** - The **Project**, **Component Dependencies**, **Resource Hierarchy/Dependencies**, **Outline**, **Attributes**, **Model**, **XSLT/XQuery input**, **XPath Builder**, and **Transformation Scenarios** views are shown.
  - **Custom** - Allows you to specify a custom layout to be used. You can save your preferred layout using **Window > Export Layout ...**, then enter the location of the saved layout file in this setting.
  - **Reset layout at startup** - When this option is enabled, Oxygen XML Editor forgets any changes made to the layout during a session and reloads the default layout the next time it is started. This is useful when you want to keep a fixed layout from one session to another.
  - **Remember layout changes for each project** - When this option is enabled, Oxygen XML Editor saves layouts individually for each project. When you switch projects, the layout you last used for that project is loaded automatically.
- **Allow detaching editors from main window** - When this option is enabled, you can drag and drop an editor window outside of the main screen. This is useful especially when you are using two monitors and you want to view files side by side.



**Note:** If the main screen is maximized, you cannot drag and drop an editor outside of it.

- **View tab placement** - This option specifies whether the *View* tabs are located at the top or bottom of the window.
- **Editor tab placement** - This option specifies whether the *Editor* tabs are located at the top or bottom of the window.

The changes you make to any layout are preserved between working sessions. The predefined *layout* files are saved in the preferences directory of Oxygen XML Editor.

To watch our video demonstration about configuring the user interface of Oxygen XML Editor, go to [http://oxygentools.com/demo/Dockable\\_Views.html](http://oxygentools.com/demo/Dockable_Views.html).

## Encoding Preferences

Oxygen XML Editor lets you configure how character encodings are recognized when opening files and which encodings are used when saving files. To configure encoding options, [open the Preferences dialog box](#) and go to **Encoding**. The following encoding options are available:

- **Fallback character encoding** - Default character encoding of non-XML documents if their character encoding cannot be determined from other sources (like, for example, specified in the document itself, or determined by the file type).



**Note:** For certain document types, the following encoding detection rules are used:

- For XML, DTD and CSS documents, Oxygen XML Editor tries to collect the character encoding from the document. If no such encoding is found, then *UTF-8* is used.
- For JavaScript, JSON, SQL, XQuery, and RNC, the *UTF-8* encoding is used.
- **UTF-8 BOM handling** - This setting specifies how to handle the *Byte Order Mark* (BOM) when Oxygen XML Editor saves a UTF-8 XML document:
  - **Don't Write** - do not save the BOM bytes. Loaded BOM bytes are ignored.
  - **Write** - save the BOM bytes.
  - **Keep** - do not alter the BOM declaration of the currently open file. This is the default option.
- **Note:** The UTF-16 BOM is always preserved. UTF-32 documents have a *big-endian* byte order.
- **Encoding errors handling** - This setting specifies how to handle characters that cannot be represented in the character encoding that is used when the document is opened. The available options are:
  - **REPORT** - displays an error identifying the character that cannot be represented in the specified encoding. Unrecognized characters are rendered as an empty box. This is the default option.
  - **IGNORE** - the error is ignored and the character is not included in the document displayed in the editor.

**Attention:** If you edit and save the document, the characters that cannot be represented in the specified encoding are dropped.

  - **REPLACE** - the character is replaced with a standard replacement character. For example, if the encoding is UTF-8, the replacement character has the Unicode code FFFD, and if the encoding is ASCII, the replacement character code is 63.

## Editor Preferences

Oxygen XML Editor lets you configure how the editor appears. To configure the appearance of the text editor, [open the Preferences dialog box](#) and go to **Editor**.

The following options are available:

- **Selection background color** - Sets the background color of selected text.
- **Selection foreground color** - Sets the text color of selected text.
- **Completion proposal background** - Sets the background color of the content completion window.
- **Completion proposal foreground** - Sets the foreground color of the content completion window.
- **Documentation window background** - Sets the background color of the documentation of elements suggested by the content completion assistant.
- **Documentation window foreground** - Sets the foreground color for the documentation of elements suggested by the content completion assistant.
- **Find highlight color** - Sets the color of the highlights generated by the **Find** and **Find all** actions.
- **XPath highlight color** - Sets the color of the highlights generated when you run an XPath expression.
- **Declaration highlight color** - Sets the color of the highlights generated by the **Find declaration** action.
- **Reference highlight color** - Sets the color of the highlights generated by the **Find reference** action.
- **Maximum number of highlights** - Sets the maximum number of highlights that Oxygen XML Editor displays.
- **Show TAB/NBSP/EOL/EOF marks** - Makes the TAB/NBSP/EOL/EOF characters visible in the editor. You can use the color picker to choose the color of the marks.
  - **Show SPACE marks** - Makes the space character visible in the editor.
- **Can edit read only files** - If this option is selected, Oxygen XML Editor will let you edit, but not save, a read only file. If the option is not selected, you cannot edit or save a read only file.
- **Display quick-assist and quick-fix side hints** - Displays the Quick Assist and Quick Fix icon in the editor's left side line number stripe. Works both in the **Text** and **Author** edit modes.

- **Undo history size** - Sets the maximum amount of undo operations you can perform in either of the editor modes (Text, Author, Design, Grid).

## Print Preferences

Oxygen XML Editor lets you configure how files are printed out of the editor. Note that these setting cover how files are printed directly from Oxygen XML Editor itself, not how they are printed after the XML source has been transformed by a publishing stylesheet. To configure the **Print** options, [open the Preferences dialog box](#) and go to **Editor > Print**.

This page allows you to customize the headers and footers added to a printed page when you print from the **Text** mode or **Author** mode editors. These settings do not apply to the **Grid** and schema **Design** modes.

You can specify what is printed on the **Left**, **Middle**, and **Right** of the header and footer using plain text of any of the following variables:

- `${currentFileURL}`  - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- `${cfne}`  - Current file name with extension. The current file is the one currently opened and selected.
- `${cp}`  - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- `${tp}`  - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor / Print** Preferences page.
- `${env(VAR_NAME)}`  - Value of the `VAR_NAME` environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the  `${system(var.name)}`  editor variable.
- `${system(var.name)}`  - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the  `${env(VAR_NAME)}`  editor variable instead.
- `${date(pattern)}`  - Current date. The allowed patterns are equivalent to the ones in the [Java SimpleDateFormat class](#).

**Example:**  `YYYY-MM-dd;`



**Note:** This editor variable supports both the `xs:date` and `xs:datetime` parameters. For details about `xs:date`, go to <http://www.w3.org/TR/xmlschema-2/#date>. For details about `xs:datetime`, go to <http://www.w3.org/TR/xmlschema-2/#dateTime>.

For example, to show the current page number and the total number of pages in the top right corner of the page, write the following pattern in the **Right** text area of the **Header** section:  `${cp} of ${tp}`.

You can also set the **Color** and **Font** used in the header and footer. Default font is SansSerif.

You can place a line below the header or above the footer by selecting **Underline/Overline**.

## Edit modes Preferences

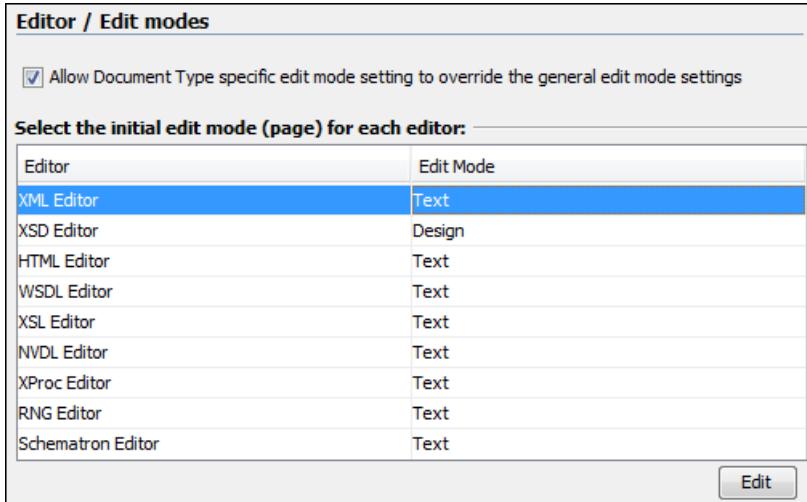
Oxygen XML Editor lets you configure which [edit mode](#) a file is opened in the first time it is opened. This setting only affects the first time a file is opened. The current editing mode of each file is saved when the file is closed and restored the next time it is opened. To configure the **Edit modes** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes**.

If **Allow Document Type specific edit mode setting to override the general mode setting** is selected, the initial edit mode setting set in [the Document Type dialog](#) overrides the general edit mode setting from the table below.

The initial edit mode can be one of the following:

- [Text](#)
- [Author](#)
- [Grid](#)
- Design (available only for the W3C XML Schema editor).

The Oxygen XML Editor “Edit modes” Preferences Page



## Text Preferences

Oxygen XML Editor lets you configure how [text mode](#) editor appears. To configure the Text mode editor options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Text**.

The following preferences are available:

- **Editor background color** - Sets the background color of the **Text** editing mode, **Diff Files** editors and **Outline** view.
- **Editor caret color** - Sets the color of the caret.
- **Highlight current line** - Sets the foreground color of the line numbers displayed in the editor panels.
- **Show line numbers** - Shows line numbers in the editor panels and in the **Results** view of the Debugger perspective.
- **Show print margin** - In conjunction with the **Print margin column** option, allows you to set a safe print limit in the form of a vertical line displayed in the right side of the editor pane. You can also customize the print margin line color.
  - **Print margin column** - Safe print limit width measured in characters.
- **Line wrap** - Enables *soft wrap* of long lines, that is automatically wrap lines in edited documents. The document content is unaltered as the application does not use newline characters to break long lines.



**Note:** When you enable the **Line wrap** option, Oxygen XML Editor disables the **Highlight current line** option.

- **Cut / Copy whole line when nothing is selected** - Enables the *Cut* and *Copy* actions when nothing is selected in the editor. In this case the *Cut* and *Copy* actions operate on the entire current line.
- **Enable folding** - Displays the vertical stripe that holds the [folding markers](#).
- **Highlight matching tag** - If you place the cursor on a start or end tag, Oxygen XML Editor highlights the corresponding member of the pair. You can also customize the highlight color.
- **Lock the XML tags** - Default tag locking state of the opened editors. For more information, see the [Locking and Unlocking XML markup](#) section.

## Diagram Preferences

For certain XML languages, Oxygen XML Editor provides a diagram view as part of the text mode editor. To configure the **Diagram** preferences, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Text / Diagram**.

The following options are available:

- **Show Full Model XML Schema diagram** - When this option is selected, the **Text** mode editor for XML Schemas is shown with a split screen view which shows a diagram of the schema structure. This may be useful for seeing the effect of schema changes you make in text view. For editing a the schema using diagram rather than text, use the [schema Design view](#).



**Note:** When handling very large schemas, displaying the schema diagram might affect the performance of your system. In such cases, disabling the schema diagram view improves the speed of navigation through the edited schema.

- **Enable Relax NG diagram and related views** - Enables the Relax NG schema diagram and synchronization with the related views (**Attributes**, **Model**, **Elements**, **Outline**).
- **Show Relax NG diagram** - Displays the Relax NG schema diagram in **Full Model View** and **Logical Model View**.
- **Enable NVDL diagram and related views** - Enables the NVDL schema diagram and synchronization with the related views (**Attributes**, **Model**, **Elements**, **Outline**).
- **Show NVDL diagram** - Displays the NVDL schema diagram in **Full Model View** and **Logical Model View**.
- **Location relative to editor** - Sets the location of the schema diagram panel relative to the diagram **Text** editor.

### Grid Preferences

Oxygen XML Editor provides a *Grid view* of an XML document. To configure the **Grid** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Grid**.

The following options are available:

- **Compact representation** - If selected, the *compact representation* of the grid is used: a child element is displayed beside the parent element. In the *non-compact representation*, a child element is nested below the parent.
- **Format and indent when passing from grid to text or on save** - If selected, the content of the document is *formatted and indented* each time you switch from the **Grid** view to the **Text** view.
- **Default column width (characters)** - Sets the default width in characters of a table column of the grid. A column can hold:
  - element names
  - element text content
  - attribute names
  - attribute values

If the total width of the grid structure is too large you can resize any column by dragging the column margins with the mouse pointer, but the change is not persistent. To make it persistent, set the new column width with this option.

- **Active cell color** - Sets the background color for the active cell of the grid. There is only one active cell at a time. The keyboard input always goes to the active cell and the selection always contains it.
- **Selection color** - Background color for the selected cells of the grid except the active cell.
- **Border color** - The color used for the lines that separate the grid cells.
- **Background color** - The background color of grid cells that are not selected.
- **Foreground color** - The text color of the information displayed in the grid cells.
- **Row header colors - Background color** - The background color of row headers that are not selected.
- **Row header colors - Active cell color** - The background color of the row header cell that is currently active.
- **Row header colors - Selection color** - The background color of the header cells corresponding to the currently selected rows.
- **Column header colors - Background color** - The background color of column headers that are not selected.
- **Column header colors - Active cell color** - The background color of the column header cell that is currently active.
- **Column header colors - Selection color** - The background color of the header cells corresponding to the currently selected columns.

The column headers are painted with two color gradients, one for the upper 1/3 part of the header and the other for the lower 2/3 part. The start and end colors of the first gradient are set with the first two color buttons. The start and end colors of the second gradient are set with the last two color buttons.

### Author Preferences

Oxygen XML Editor provides an *author mode* editor which provides a configurable graphical editing interface to XML documents. To configure the **Author** mode preferences, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author**.

The following options are available:

- **Author default background color** - Sets the default background color of the Author editing mode. The `background-color` property set in the CSS file associated with the current edited document overwrites this option.
- **Author default foreground color** - Sets the default foreground color of the Author editing mode. The `color` property set in the CSS file associated with the current edited document overwrites this option.
- **Show XML comments** - When this option is selected, XML comments are displayed in **Author** mode, otherwise they are hidden.
- **Show processing instructions** - When this option is selected, XML processing instructions are displayed in **Author** mode, otherwise they are hidden.
- **Show doctype** - When this option is selected, the `doctype` declaration is displayed in **Author** mode; otherwise it is hidden.
- **Show placeholders for empty elements** - When this option is selected, placeholders are displayed for elements with no content to make them clearly visible. The placeholder is rendered as a light grey box displaying the element name.
- **Show Author layout messages** - When this option is selected, all errors reported while rendering the document in Author mode are presented in the **Errors** view.
- **Show block range** - When this option is selected, a *block range indicator* is shown in a stripe located in the left side of the editor. The block range indicator is displayed as a heavy line that spans from the first line to the last line of the block.
- **Display referenced content (e.g.: entities, XInclude, DITA conref, etc.)** - when enabled, the references (like entities, XInclude, DITA conref) also display the content of the resources they reference. If you toggle this option while editing, reload the file for the modification to take effect.
- **Auto-scale images wider than (pixels)** - Sets the maximum width at which an image will be displayed. Wider images will be scaled to fit.
  - **Show very large images** - When this option is selected, images larger than 6 megapixels are displayed in **Author** mode, otherwise they are not displayed.



**Important:** If you enable this option and your document contains many such images, Oxygen XML Editor may consume all available memory, throwing an *OutOfMemory error*. To resolve this, increase the [available memory limit](#), and restart the application.

- **Format and indent** - here you can set the format and indent method that is applied when a document is saved in **Author** mode, or when switching the editing mode (from Text to Author or vice versa):
  - **Only the modified content** - the **Save** operation only formats the nodes that were modified in the **Author** mode. The rest of the document preserves its original formatting.



**Note:** This option applies also to the DITA Maps open in the **DITA Maps Manager**.

- **The entire document** - the **Save** operation applies formatting to the entire document regardless of the nodes that were modified in **Author** mode.

If the **Apply also 'Format and Indent' action as in 'Text' edit mode** option is enabled, the content of the document is formatted by applying the **Format and Indent** rules from the [Editor/Format/XML](#) option page. In this case, the result of the **Format and indent** operation will be the same as when it is applied in **Text** editing mode.

- **Tags display mode** - Sets the default display mode for element tags presented in **Author** mode. You can choose between:
  - **Full Tags with Attributes** - All XML tags are shown, with attribute names and values, easing the transition from a Text based editing to an **Author** mode editing.
  - **Full Tags** - All XML tags are shown, but without attributes.

- **Block Tags** - The XML tags that enclose block elements are shown in full. Compact tags (no element names) are shown for inline elements.
- **Inline Tags** - The XML tags that enclose inline elements are shown in full. Block tabs are not shown.
- **Partial Tags** - Partial tags (no names) are shown for all elements.
- **No Tags** - No tags are displayed. This representation is as close as possible to a word-processor view.
- **Tags background color** - Sets the Author tags background color.
- **Tags foreground color** - Sets the Author tags foreground color.
- **Tags font** - Allows you to change the font used to display tags text in the **Author** visual editing mode. The *[Default]* font is computed based on the setting of the [Author default font option](#).
- **Compact tag layout** - When you deselect this option, the Author mode displays the tags in a more decompressed layout, where block tags are displayed on separate lines.

### Caret Navigation Preferences

Oxygen XML Editor allows you to configure the appearance of the *caret* (text cursor) in the [author mode](#) editor. To set caret navigation preferences, [open the Preferences dialog box](#) and go to **Author > Caret Navigation**. The following options are available:

- **Highlight elements near caret** - When this option is selected, the element containing the caret is highlighted. You can use the color picker to choose the color of the highlight.
- **Show caret position tooltip** - Oxygen XML Editor uses [tool tips in Author mode to indicate the position of the caret in the element structure](#) of the underlying document. Depending on context, the tool tips may show the current element name or the names of the elements before and after the current caret position.
- **Show location tooltip on mouse move** - When this option is selected, Oxygen XML Editor displays [Location Tooltips](#) when you are editing the document in certain tags display modes (Inline Tags, Partial Tags, No Tags) and the mouse pointer is moved between block elements.
- **Quick up/down navigation** - By default, when you navigate using the up and down arrow keys in Author mode, the caret is placed within each of the underlying XML elements between two blocks of text. (The caret turns horizontal when it is between blocks of text.) For instance, between a list item in one section and the title in a following sections, the caret might stop several times in the underlying structure: the list item, the list, the paragraph, the section, and the root element between sections, the new section, and finally in the title. Any one of these location is a place you might want to insert new content. When this option is selected, however, the caret does not stop at these positions, but jumps from one text line to another, similar to how the caret behaves in a word processor.
- **Arrow keys move the caret in the writing direction** - This setting determines how the left and right arrow keys behave in Author mode for bidirectional (BIDI) text. When this option is selected, the right arrow key advances the caret in the reading direction. When this option is not selected, pressing the right arrow will simply move the caret to the right, regardless of the text direction.

### Schema-Aware Preferences

Oxygen XML Editor can use the schema of your XML language to improve the way the [Author mode](#) editor handles your content. To configure the **Schema Aware** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Schema aware**.

- **Schema aware normalization, format, and indent**

When you open or save a document in **Author** mode, white space is normalized using the *display* property of the current CSS stylesheet and the values of the *settings* for **Preserve space elements**, **Default space elements**, and **Mixed content elements**. When this option is selected, the schema will also be used to normalize white space, based on the content model (*element-only*, *simple-content*, or *mixed*) . Note that the schema information takes precedence.

- **Indent blocks-only content**

To avoid accidentally introducing inappropriate white space around inline elements, Oxygen XML Editor does not normally apply indenting to the source of an element with mixed content. If this option is selected, Oxygen XML Editor will apply indenting to the source of mixed content elements that only contain block elements.

- **Schema Aware Editing**

This setting determines how Oxygen XML Editor will use the schema of a document to control the behavior of the **Author** mode.

- **On** - Enables all schema-aware editing options.
- **Off** - Disables all schema-aware editing options.
- **Custom** - Allows you to select custom schema-aware editing options from the following:

- **Delete element tags with backspace and delete**

Controls what happens when you attempt to delete an element tag. The options are:

- **Smart delete**

If deleting the tag would make the document invalid, Oxygen XML Editor will attempt to make the document valid by unwrapping the current element or by appending it to an adjacent element where the result would be valid. For instance, if you delete a bold tag, the content can be unwrapped and become part of the surrounding paragraph, but if you delete a list item tag, the list item content cannot become part of the list container. However, the content could be appended to a preceding list items.

- **Reject action when its result is invalid**

A deletion that would leave the document in an invalid state is rejected.

- **Paste and Drag and Drop**

Controls the behavior for paste and drag and drop actions. Available options are:

- **Smart paste and drag and drop**

If the content inserted by a paste or drop action is not valid at the caret position, according to the schema, Oxygen XML Editor tries to find an appropriate insert position. The possibilities include:

- Creating a sibling element that can accept the content. (For example, if you tried to paste a paragraph into an existing paragraph.)
- Inserting the content into a parent or child element. (For example, if you tried to paste a list item into an existing list item, or into the space above or below an existing list.)
- Inserting the content into an ancestor element where it would be valid.

- **Reject action when its result is invalid**

If this option is enabled and the **Smart paste and drag and drop** option is disabled, Oxygen XML Editor will not let you paste content into a position where it would be invalid.

- **Typing**

Controls the behavior that takes place when typing. Available options:

- **Smart typing**

If typed characters are not allowed in the element at the caret position, but the previous element does allow text, then a similar element will be inserted, along with your content.

- **Reject action when its result is invalid**

If checked, and the result of the typing action is invalid, the action will not be performed.

- **Content Completion**

Controls the behavior that takes place when inserting elements using content completion. Available options are:

- **Allow only insertion of valid elements and attributes**

If selected, the content completion list shows only the elements that can be inserted at the current position and will not allow you to enter any other element.

- **Show all possible elements in the content completion list**

If selected, the content completion list will show all the elements in the schema, even those that cannot be entered validly at the current position. If you select an element that is not valid at the current position, Oxygen XML Editor will attempt to find a valid location to insert it and may present you with several options.

- **Warn on invalid content when performing action**

A warning message will be displayed when performing an action that will result in invalid content. Available options are:

- **Delete Element Tags**

If selected, a warning message will be displayed if the [Delete Element Tags](#) action will result in an invalid document. You will be asked to confirm the deletion.

- **Join Elements**

If selected, a warning message will be displayed if the [Join Elements](#) action will result in an invalid document. You will be asked to confirm the join.

- **Convert external content on paste**

If selected, turns on [smart paste](#) for external content.

### *Review Preferences*

Oxygen XML Editor lets you [enter review comments and track changes](#) in your documents. The Review preferences control how the Oxygen XML Editor review features work. To configure the **Review** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Review**.

The available options are:

- **Author** - Specifies the name to be attached to all comments and to changes made while **Track Changes** is active. By default, Oxygen XML Editor uses the system user name.
- **Initial State** - Specifies whether or not **Track Changes** is enabled when you open a document. You may have some opened documents in which track changes is enabled and others in which it is disabled. You can choose between the following options:
  - **Stored in document** - The current state of track changes is stored in the document itself, meaning that track changes on or off depending on the state the last time the document was saved. This is the recommended setting when multiple authors work on the same set of documents as it will make it obvious to other authors that changes have been made in the document.
  - **Always On** - The **Track Changes** feature is always on when you open a document. You can turn it off for an open document, but it will be turned on for the next document you open.
  - **Always Off** - The **Track Changes** feature is always off when you open a document. You can turn it on for an open document, but it will be turned off for the next document you open.
- **Display changed lines marker** - A changed line maker is a vertical line on the left side of the editor window indicating where changes have been made in the document. To hide the changed lines marker, deselect this option.
- **Inserted content color** - When **Track Changes** option is on, the newly inserted content is highlighted with an *insertion marker*, that uses a color to adjust the following display properties of the inserted content: foreground, background, and underline. This section allows you to customize the marker's color:
  - **Automatic** - Oxygen XML Editor assigns a color to each user who inserted content in the current document. The colors are picked from the **Colors for automatic assignment** list, the priority being established by the change (deletion, insertion, or comment) timestamp.
  - **Fixed** - Uses the specified color for all insertion markers, regardless of who the author is.

- **Use same color for text foreground** - Use the color defined above (**Automatic** or **Fixed**) to render the foreground of the inserted content.
- **Use same color for background** - Use the color defined above (**Automatic** or **Fixed**) to render the background of the inserted content. A slider control allows you to set the transparency level of the marker's background.
- **Deleted content color** - When **Track Changes** option is on, the deleted content is highlighted with a *deletion marker*, that uses a color to adjust the following display properties of the deleted content: foreground, background, and strikethrough. This section allows you to customize the marker's color:
  - **Automatic** - Oxygen XML Editor assigns a color to each user who deleted content in the current document. The colors are picked from the **Colors for automatic assignment** list, the priority being established by the change (deletion, insertion, or comment) timestamp.
  - **Fixed** - Uses the specified color for all deletion markers, regardless of who the author is.
  - **Use same color for text foreground** - Use the color defined above (**Automatic** or **Fixed**) to render the foreground of the deleted content.
  - **Use same color for background** - Use the color defined above (**Automatic** or **Fixed**) to render the background of the deleted content. A slider control allows you to set the transparency level of the marker's background.
- **Comments color (applies for all authors)** - Sets the background color of the text that is commented on. The options are:
  - **Automatic** - Oxygen XML Editor assigns a color to each user who added a comment in the current document. The colors are picked from the **Colors for automatic assignment** list, the priority being established by the change (deletion, insertion, or comment) timestamp.
  - **Fixed** - Uses the specified color for all changes, regardless of the author's name. Use the slider to control the transparency level.

## Callouts Preferences

Oxygen XML Editor can display callouts for *review items such as comment, insertion, and deletions*. To set review callouts preferences, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Review > Callouts**.

The options are:

- **Comments** - If selected, callouts are shown for comments. This option is enabled by default.
- **Track Changes deletion** - If selected, callouts are shown for deletions.
- **Show deleted content in callout** - If selected, the deleted content is shown in the callout.
- **Track Changes insertion** - If selected, callouts are shown for insertions.
- **Show inserted content in callout** - If selected, the inserted content is shown in the callout.
- **Show review time** - When selected, the date and time of a change are shown in the callout.
- **Show all connecting lines** - When selected, lines connect the callout to the location of the change.
- **Callouts pane width (px)** - Sets the width of the callout field. The default is 200 pixels.
- **Callouts text limit (characters)** - Sets the number of characters shown in the callout. The default is 160. Note that this does not limit the number of characters in a comment. It only limits the number of characters shown in the callout. To see the full comment, see the [review view](#).

## Profiling / Conditional Text Preferences

Oxygen XML Editor lets you configure how *profiling and conditional text* is shown in Author view. It has built in support for the standard conditional text features of DITA and DocBook, which you can customize for your own projects. You can also add conditional support for other XML vocabularies, including your custom vocabularies.

To configure Profiling/Conditional Text options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Profiling/Conditional Text**.



**Note:** Please note the following when configuring this setting:

- This setting is used to define how profiled elements are treated in Author view. It does not create profiling or conditional text attributes or values in the underlying XML vocabulary. It just changes how the editor displays them.
- This setting should be used for profiling / conditional text elements only. To change how other types of attributes are displayed in the text, use a CSS style sheet.
- If you are using the DITA XML vocabulary and a DITA Subject Scheme Map is defined in the root map of your document, it will be used in place of anything defined using this dialog.

This preferences page contains two items:

- **profiling attributes**, which allow to specify a set of allowable value for each profiling or conditional attribute.
- **profiling condition sets** which allow you to specify a specific set of profiling attributes to be used to specify a particular build configuration for your content.

If you have two or more identically named entries that match the same document type, Oxygen XML Editor uses the one that is positioned highest in the table. Use the **Up** / **Down** buttons to change the priority of the entries.

The **Import from DITAVAL** button allows you to import profiling attributes from `.ditaval` files. You can merge these new profiling attributes with the existing ones, or replace them completely. If the imported attributes conflict with the existing ones, Oxygen XML Editor displays a dialog box containing two tables: the first one previews the imported attributes and the second one previews the already defined attributes. You can choose either to keep the existing attributes or replace them with the imported ones.



**Note:** When importing profiling attributes from DITAVAL files, Oxygen XML Editor automatically creates condition sets based on these files.

## Colors and Styles Preferences

Oxygen XML Editor lets you set the colors and styles used to display *profiling / conditional text* in the *Author mode editor*. To set Colors and Styles preferences, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Profiling/Conditional Text > Colors and Styles**.

The central area of the page contains a table that lists two categories of profiling styles:

- **Defined attributes values** - contains the styles for profiling attribute values defined in the *Profiling / Conditional Text* preferences page. Each profiling attribute value has an associated style. To ease the process of customizing styles, the **Defined attributes values** category contains by default the list of empty styles. All you have to do is to adjust the colors and decorations, thus skipping the process of manually defining the association rules (document type, attribute name and value). This is the reason why a style from this category can only be *reset*, not deleted.
- **Other** - this category contains styles for attribute values that are not marked as profiling values, in the *Profiling / Conditional Text* preferences page. In this category are listed:
  - all the styles that were defined in other projects (with different profiling attribute value sets)
  - all the styles set for the profiling attributes defined in a *subject scheme map*

### Adding a profiling style

To add profiling styles use one of the following actions:

#### Import from DITAVAL...

Allows you to import profiling styles from `.ditaval` files. You can merge these new profiling styles with the existing ones, or replace them completely. If the imported styles conflict with the existing ones, Oxygen XML Editor displays a dialog box containing two tables: the first one previews the imported styles and the second one previews the already defined styles. You can choose either to keep the existing styles or replace them with the imported ones.

#### Automatic styling

For every profiling attribute value that has no style defined, applies one of the following color or style: background, foreground, text decoration (underline, overline, double-underline) or text style (bold, italic).

## New

Opens the **Add Profiling Style** dialog box that allows you to associate a set of coloring and styling properties to a profiling value.



**Note:** You can define a default style for a specific attribute by setting the **Attribute value** field to <ANY>. This style is applied for attribute values that do not have a specific style associated with it.

## Modify a profiling style

To modify an previously defined style, use one of the following actions:

- Double-click the style in the styles table to open the **Edit Profiling Style** dialog box.
- Select the style in the styles table and press **Edit** to open the **Edit Profiling Style** dialog box.

## Resetting a profiling style from Defined attributes values category

To reset a style from the **Defined attributes values** category to its default (no color or decoration), select it and click the **Clear style** button.

## Deleting a profiling style from Other category

To delete a style from the **Other** category, select it and click the **Delete** button.

### Attributes Rendering Preferences

Oxygen XML Editor lets you *display the profiling attributes applied to your content* in the Author mode editor. To configure how the profiling attributes appear, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > Profiling/Conditional Text > Attributes Rendering**. When the **Show Profiling Attributes** option is enabled, the Author page displays conditional text markers at the end of conditional text blocks. Use the options in this page to customize the rendering of these text markers.

You can set the following options:

- **Show profiling attribute name** - If checked, the names of the profiling attributes are shown with their values. If unchecked, only the values are shown.
- **Background color** - Sets the background color used to display the profiling attributes.
- **Attribute name foreground color** - Sets the foreground color used to display the names of the profiling attributes.
- **Attribute values foreground color** - Sets the foreground color used to display values of the profiling attributes.
- **Border color** - Sets the color of the border of the block that displays the profiling attributes.

### MathML Preferences

Oxygen XML Editor allows you to [edit MathML](#) equations and displays the results in a preview window. For a more specialized *MathML* editor, you can [install Design Science MathFlow](#), which is a commercial product that requires a separate license.

To configure the *MathML* editor or to enter your *MathFlow* license information, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Author > MathML**. You can configure the following parameters:

- **Equation minimum font size** - The minimum size of the font used for rendering mathematical symbols when editing in the **Author** mode.

The following options can be configured for *MathFlow*:

- **MathFlow installation directory** - The installation folder for the *MathFlow Components* product (*MathFlow SDK*).
- **MathFlow license file** - The license for *MathFlow Components* product (*MathFlow SDK*).
- **MathFlow preferred editor** - A *MathML* formula can be edited in one of three editors of *MathFlow Components* product(*MathFlow SDK*):
  - **Structure Editor** - (default selection) targets professional XML workflow users
  - **Style Editor** - tailored to the needs of content authors

- **Simple Editor** - designed for applications where end-users can enter mathematical equations without prior training and only the meaning of the math matters
- **Save special characters** - When editing mathematical expressions the special characters can be saved in the XML file:
  - **As entity names** - saves the characters in &name ; format. It refers to a character by the name of the entity which has the desired character as its replacement text. For example, the Greek *Omega* character is saved as &Omega ;.
  - **As character entities** (default selection) - saves the characters in a hexadecimal value, using the &#xNNN format. For example, the Greek *Omega* character is saved as &#x3a9 ;.
  - **As character values** - saves the characters as the actual symbol. For example, the Greek *Omega* character is saved as .

More documentation is available on the [Design Science MathFlow](#) website.

#### *AutoCorrect Preferences*

Oxygen XML Editor includes an option to automatically correct misspelled words as you type in **Author** mode. To enable and configure this feature, [open the Preferences dialog box](#) and go to **Editor > Edit Modes > Author > AutoCorrect**.

The following options are available:

- **Enable AutoCorrect** - This option is disabled by default. If enabled, while editing in **Author** mode, if you type anything that is listed in the **Replace** column of the Replacements table displayed in this preferences page, Oxygen XML Editor will automatically replace it with the value listed in the **With** column.
- **Use additional suggestions from the spell checker** - If enabled, in addition to anything listed in the Replacements table displayed in this preferences page, Oxygen XML Editor will also use suggestions from the Spell Checker to automatically correct misspelled words. Suggestions from the Spell Checker will only be used if the misspelled word is not found in the Replacements table.

 **Note:** The *AutoCorrect* feature shares the same options configured in the [Language options](#) and [Ignore elements](#) sections in the [Spell Check](#) preferences page.

#### **Replacements Table Section**

The *AutoCorrect* feature uses the Replacements table to automatically replace anything that is listed in the **Replace** column with the value listed in the **With** column for each language. You can specify the language in the **Replacements for language** drop-down list, and for each language, you can configure the items listed in the table. The language selected in this page is not the language that will be used by the *AutoCorrect* feature. It is simply the language for which you are configuring the Replacements table.

 **Note:** Any changes, additions, or deletions you make to this table are saved to a path that is specified in the [AutoCorrect Dictionaries preferences page](#).

#### **Smart Quotes Section**

You can also choose to automatically convert double and single quotes to a quotation characters of your choice by using the following options in the **Smart quotes** section:

- **Replace "Single quotes"** - Replaces single quotes with the quotation symbols you select with the **Start quote** and **End quote** buttons.
- **Replace "Double quotes"** - Replaces double quotes with the quotation symbols you select with the **Start quote** and **End quote** buttons.

#### **Global and Project Options Section**

Selecting **Project Options** in this preferences page will only save your selections in **Enable AutoCorrect**, **Use additional suggestions from the spell checker**, and the options in the **Smart quotes** section. Changes to the Replacements table are not saved in this page. To save changes to the Replacements table at project level you need to specify a custom

location in [the User-defined replacements section of the AutoCorrect Dictionaries preferences page](#) and select **Project Options** from that preferences page instead.

**Restore Defaults** - Restores the options in this preferences page to their default values and also **deletes any changes you have made to the Replacements table**.

#### AutoCorrect Dictionaries Preferences

To set the Dictionaries preferences for the *AutoCorrect* feature, [open the Preferences dialog box](#) and go to **Editor > Edit Modes > Author > AutoCorrect > Dictionaries**. This page allows you to specify the location of the dictionaries that Oxygen XML Editor uses for the *AutoCorrect* feature and the location for saving user-defined replacements.

The following options are available in this preferences page:

- **Dictionaries default folder** - Displays the default location where the dictionaries that Oxygen XML Editor uses for the *AutoCorrect* feature are stored.
- **Include dictionaries from** - Enable this option if you want to specify an additional location for the dictionaries that Oxygen XML Editor will use for the *AutoCorrect* feature.



**Note:** The *AutoCorrect* feature takes into account dictionaries collected both from the default and custom locations and multiple dictionaries from the same language are merged into a generic dictionary (for example, `en_UK.dat` from the default location is merged with `en_US.dat` from a custom location, and the result is that a third file is created for a generic dictionary called `en.dat`). However, if there is already a generic dictionary (for example, `en.dat`) saved in either the default or custom location, the other specific dictionaries (for example, `en_UK.dat` and `en_US.dat`) will not be merged and the existing generic dictionary will simply be used. Also, if the additional location contains a file with the same name as one from the default location, the file in the additional location takes precedence over the file from the default location. The user-defined replacements are never merged.

- **Save user-defined replacements in the following location** - Specifies the target where added, edited, or deleted replacements are saved. By default, the target is the application preferences folder, but you can also choose a custom location.



**Tip:** To save changes to [the Replacement table \(in the AutoCorrect preferences page\)](#) at project level, select a custom location for the **User-defined replacements** and select **Project Options** at the bottom of the page.

#### Schema Design Preferences

Oxygen XML Editor provides a [graphical schema design editor](#) to make editing XML schemas easier. To configure the **Schema Design** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Schema Design**

The following options are available in the **Schema Design** preferences page:

- **Show annotation in the diagram** - When selected, Oxygen XML Editor displays the content of `xs:documentation` elements in the XML Schema **Design** view.
- **When trying to edit components from another schema** - The schema diagram editor will combine schemas imported by the current schema file into a single schema diagram. You can choose what happens if you try to edit a component from an imported schema. The options are:
  - **Always go to its definition** - Oxygen XML Editor opens the imported schema file so that you can edit it.
  - **Never go to its definition** - The imported schema file is not opened. The definition cannot be edited in place.
  - **Always ask** - Oxygen XML Editor asks if you want to open the imported schema file.
- **Zoom** - Allows you to set the default zoom level of the schema diagram.

#### Properties

Oxygen XML Editor lets you control which properties to display for XML Schema components in the [XML Schema Design view](#). To configure the schema design properties displayed, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Schema Design > Properties**.

The available options are:

- **Show additional properties in the diagram** - If selected, the properties selected in the property table are shown in the XML Schema Diagram view. This option is selected by default.
- The properties that can be selected. In the table, select those properties you want to be displayed. You can also select if you want the property to be displayed only when it is actually defined in the schema.

## Format Preferences

This preferences page contains various formatting options which influence editing and formatting both in the **Text** and **Author** modes. To control additional options specifically for the *Author mode* editor, see *Author Whitespace Handling* on page 148.

-  **Note:** These settings apply to the formatting of source documents. The formatting of output documents is determined by the *transformation scenarios that create them*.

The following options are available:

- **Detect indent on open** - Oxygen XML Editor detects how a document is indented when it is opened. Oxygen XML Editor uses a heuristic method of detection by computing a weighted average indent value from the initial document content. You can disable this setting if the detected value does not work for your particular case and you want to use a fixed-size indent for all the edited documents.
-  **Tip:** If you want to minimize the formatting differences created by the **Format and Indent** operation in a document edited in the **Text** edited mode, make sure that both the **Detect indent on open** and **Detect line width on open** options are enabled.
  - **Use zero-indent, if detected** - by default, if no indent was detected in the document, the fixed-size indent is used. Enable this option if all your document have no indentation and you want to keep them that way.
- **Indent with tabs** - If selected, indents are created using tab characters. If unchecked, lines are indented using space characters.
- **Indent size** - A fixed number of spaces used for indenting a line.
- **Hard line wrap (Limit to "Line width - Format and Indent")** - If selected, when typing content in the **Text** editing mode when the maximum line width is reached, a line break is automatically inserted.
- **Indent on enter** - If disabled, when you press the `Enter` key to insert a line break in the **Text** editing mode, no indentation will be added to the new line.
- **Enable smart enter** - If selected, when you press the `Enter` key between a start and an end XML tag in the **Text** editing mode, the cursor is placed in an indented position on the empty line formed between the start and end tag.
- **Detect line width on open** - When selected, Oxygen XML Editor detects the line width automatically when the document is opened.
- **Format and indent the document on open** - When selected, an XML document is formatted and indented before opening it in Oxygen XML Editor.
- **Line width - Format and Indent** - Defines the number of characters after which the **Format and Indent** (pretty-print) action performs hard line wrapping. For example, if set to 100, after a **Format and Indent** action, the longest line will have at most 100 characters. This setting is also used when saving the XML content edited in the **Author** editing mode.
- **Clear undo buffer before Format and Indent** - The **Format and Indent** operation can be undone, but if used intensively, a considerable amount of the memory allocated for Oxygen XML Editor will be used for storing the undo states. If this option is selected, Oxygen XML Editor empties the undo buffer before doing a **Format and Indent** operation. This means you will not be able to undo any changes you made before the format and indent operation. Select this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

The indent size and line width limit settings are used in various places in the application:

- When the **Format and Indent** action is used in the **Text** editing mode.
- When you press `ENTER` in the **Text** editing mode to break a line.
- When editing in the **Text** mode with **Hard line wrap** enabled.
- When the XML is serialized by saving content in the **Author** editing mode.

To watch our video demonstration about the formatting options offered by Oxygen XML Editor, go to [http://oxygenvxml.com/demo/Autodetect\\_Formatting.html](http://oxygenvxml.com/demo/Autodetect_Formatting.html).

## XML Formatting Preferences

To configure the **XML** Formatting options, *open the **Preferences** dialog box* and go to **Editor > Format > XML**.

The following options are available:

- **Preserve empty lines** - The **Format and Indent** operation preserves all empty lines found in the document.
- **Preserve text as it is** - The **Format and Indent** operation preserves text content as it is, without removing or adding any white space.
- **Preserve line breaks in attributes** - Line breaks found in attribute values are preserved.



**Note:** When this option is enabled, the **Break long attributes** option is automatically disabled.

- **Break long attributes** - The **Format and Indent** operation breaks long attribute values.
- **Indent inline elements** - The *inline elements* are indented on separate lines if they are preceded by white spaces and they follow another element start or end tag. Example:

Original XML:

```
<root>
 text <parent> <child></child> </parent>
</root>
```

**Indent inline elements** enabled:

```
<root> text <parent>
 <child/>
 </parent>
</root>
```

**Indent inline elements** disabled:

```
<root> text <parent> <child/> </parent> </root>
```

- **Expand empty elements** - The **Format and Indent** operation outputs empty elements with a separate closing tag (for example, `<a attr1="v1"></a>`). When not enabled, the same operation represents an empty element in a more compact form (`<a attr1="v1" />`).
  - **Sort attributes** - The **Format and Indent** operation sorts the attributes of an element alphabetically.
  - **Add space before slash in empty elements** - Inserts a space character before the trailing / and > of empty elements.
  - **Break line before attribute's name** - The **Format and Indent** operation breaks the line before the attribute name.
  - **Element spacing** - Controls how the application handles whitespaces found in XML content:
    - **Preserve space** - List of elements for which the **Format and Indent** operation preserves the whitespaces (such as blanks, tabs, and newlines). The elements can be specified by name or by XPath expressions:
      - `elementName`
      - `//elementName`
      - `/elementName1/elementName2/elementName3`
      - `//xs:localName`
- The namespace prefixes (such as `xs`) are treated as part of the element name without taking its binding to a namespace into account.
- **Default space** - The list contains the names of the elements for which the content is normalized (multiple contiguous whitespaces are replaced by a single space), before applying the **Format and Indent** operation.
  - **Mixed content** - The elements from this list are treated as mixed content when applying the **Format and Indent** operation. The lines are split only when whitespaces are encountered.
  - **Schema aware format and indent** - The **Format and Indent** operation takes into account the schema information regarding the *space preserve*, *mixed*, or *element only* properties of an element

- **Indent (when typing) in preserve space elements** - Normally, the *Preserve space* elements (identified by the `xml:space` attribute set to `preserve` or by their presence in the **Preserve space** elements list) are ignored by the **Format and Indent** operation. When this option is enabled and you edit one of these elements, its content is formatted
- **Indent on paste - sections with number of lines less than 300** - When you paste a chunk of text that has less than 300 lines, the inserted content is indented. To keep the original indent style of the document you copy content from, disable this option.

### Whitespaces Preferences

Oxygen XML Editor lets you configure which Unicode space characters are treated as space characters when normalizing whitespace in XML documents. To configure the **Whitespace** preferences, [open the Preferences dialog box](#) and go to **Editor > Format > XML > Whitespaces**.

This table lists the Unicode whitespace characters. Check any that you want to have treated as whitespace when formatting and indenting an XML document.

The whitespaces are normalized when:

- The **Format and Indent** action is applied on an XML document.
- You switch from **Text** mode to **Author** mode.
- You switch from **Author** mode to **Text** mode.

The characters with the codes 9 (TAB), 10 (LF), 13 (CR) and 32 (SPACE) are always considered to be whitespace characters and cannot be deselected.

### XQuery Formatting Preferences

To configure the **XQuery** Formatting options, [open the Preferences dialog box](#) and go to **Editor > Format > XQuery**.

The following options are available:

- **Preserve line breaks** - All initial line breaks are preserved.
- **Break line before attribute's name** - Each attribute of an XML element is written on a new line and properly indented.

### XPath Formatting Preferences

To configure the **XPath** Formatting options, [open the Preferences dialog box](#) and go to **Editor > Format > XPath**.

The following option is available:

- **Format XPath code embedded in XSLT, XSD and Schematron files** - If enabled, the **Format and Indent** action applied on a XSD, XSLT, or Schematron document will perform an XPath-specific formatting on the values of the attributes that accept XPath expressions.



**Note:** For XSLT documents, the formatting is not applied to *attribute value templates*.

### CSS Properties Formatting Preferences

Oxygen XML Editor can format and indent your CSS files. To configure the **CSS** Format options, go to **Editor > Format > CSS**.

The following options control how your CSS files are formatted and indented:

- **Indent class content** - The *class* content is indented. Enabled by default.
- **Class body on new line** - The *class* body (including the curly brackets) is placed on a new line.
- **Add new line between classes** - An empty line is added between two classes.
- **Preserve empty lines** - The empty lines from the CSS content are preserved.
- **Allow formatting embedded CSS** - The CSS content embedded in XML is formatted when the XML content is formatted.

### JavaScript Properties Formatting Preferences

To configure the **JavaScript** format options, [open the Preferences dialog box](#) and go to **Editor > Format > JavaScript**.

The following options control the behavior of the **Format and Indent** action:

- **Start curly brace on new line** - Opening curly braces start on a new line.
- **Preserve empty lines** - Empty lines in the JavaScript code are preserved. This option is enabled by default.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows Oxygen XML Editor to format embedded JavaScript code, taking precedence over the *Schema aware format and indent* option. This option is enabled by default.

## Content Completion Preferences

Oxygen XML Editor provides a **Content Completion Assistant** that lists available options at any point in a document and can auto-complete structures, elements, and attributes. These options control how the **Content Completion Assistant** works.

To configure the **Content Completion** preferences, *open the Preferences dialog box* and go to **Editor > Content Completion**.

The following options are available:

- **Auto close the last opened tag** - Oxygen XML Editor closes the last open tag when you type </>.
- **Automatically rename/delete/comment matching tag** - If you rename, delete, or comment out a start tag, Oxygen XML Editor automatically renames, deletes, or comments out the matching end tag.

 **Note:** If you select **Toggle comment** for multiple starting tags and the matching end tags are on the same line as other start tags, the end tags are not commented.

- **Use content completion** - Turns content completion on or off.
- **Close the inserted element** - When you choose an entry from the **Content Completion Assistant** list of proposals, Oxygen XML Editor inserts both start and end tags.
  - **If it has no matching tag** - The end tag of the inserted element is automatically added only if it is not already present in the document.
  - **Add element content** - Oxygen XML Editor inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is *associated with the edited XML document*.
    - **Add optional content** - Oxygen XML Editor inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
    - **Add first Choice particle** - Oxygen XML Editor inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema.
- **Case sensitive search** - When enabled, the search in the content completion assistant window when you type a character is case-sensitive ('a' and 'A' are different characters).

 **Note:** This option is ignored when the current language itself is not case sensitive. For example, the case is ignored in the CSS language.

- **Cursor position between tags** - When selected, Oxygen XML Editor automatically moves the cursor between start and end tag after inserting the element. This only applies to:
  - Elements with only optional attributes or no attributes at all.
  - Elements with required attributes, but only when the **Insert the required attributes** option is disabled.
- **Show all entities** - Oxygen XML Editor displays a list with all the internal and external entities declared in the current document when the user types the start character of an entity reference (i.e. &).
- **Insert the required attributes** - Oxygen XML Editor inserts automatically the required attributes taken from the DTD or XML Schema. This option is applied also in the **Author** mode of the XML editor.
- **Insert the fixed attributes** - Oxygen XML Editor automatically inserts any FIXED attributes from the DTD or XML Schema for an element inserted with the help of the **Content Completion Assistant**. This option is applied also in the **Author** mode of the XML editor.

- **Show recently used items** - when checked, Oxygen XML Editor remembers the last inserted items from the **Content Completion Assistant** window. The number of items to be remembered is limited by the **Maximum number of recent items shown** list box. These most frequently used items are displayed on the top of the content completion window and are separated from the rest of the suggestions by a thin grey line . This option is applied also in the **Author** mode of the XML editor.
- **Maximum number of recent items shown** - limits the number of recently used items presented at the top of the **Content Completion Assistant** window. This option is applied also in the **Author** mode of the XML editor.
- **Learn attributes values** - Oxygen XML Editor learns the attribute values used in a document. This option is applied also in the **Author** mode of the XML editor.
- **Learn on open document** - Oxygen XML Editor automatically learns the document structure when the document is opened. This option is applied also in the **Author** mode of the XML editor.
- **Learn words** (Dynamic Abbreviations, available on **Ctrl Space (Command Space on OS X)**) - When selected, Oxygen XML Editor learns the typed words and makes them available in a content completion fashion by pressing **Ctrl Space (Command Space on OS X)** om your keyboard;



**Note:** In order to be learned, the words need to be separated by space characters.

- **Activation delay of the proposals window (ms)** - Delay in milliseconds from last key press until the content completion assistant window is displayed.

### Annotations Preferences

Different types of schemas (XML Schema, DTDs, Relax NG) can include annotations that document the various elements and attributes that they define. Oxygen XML Editor can display these annotations when offering content completion suggestions. To configure the **Annotations** preferences, [open the Preferences dialog box](#) and go to **Editor > Content Completion > Annotations**.

The following options are available:

- **Show annotations in Content Completion Assistant** - Oxygen XML Editor displays the schema annotations of an element, attribute, or attribute value currently selected in the **Content Completion Assistant** proposals list.
- **Show annotations in tooltip** - Oxygen XML Editor displays the annotation of elements and attributes as a tooltip when you hover over them with the cursor in the editing area or in the **Elements** view.
- **Show annotation in HTML format, if possible** - This option allows you to view the annotations associated with an element or attribute in HTML format. It is available when editing XML documents that have associated an XML Schema or Relax NG schema. When this option is disabled the annotations are converted and displayed as plain text.
- **Prefer DTD comments that start with "doc:" as annotations** - To address the lack of dedicated annotation support in DTD documents, Oxygen XML Editor recommends prefixing with the `doc:` particle all comments intended to be shown to the developer who writes an XML validated against a DTD schema.

When this option is enabled, Oxygen XML Editor uses the following mechanism to collect annotations:

- if at least one `doc:` comment is found in the entire DTD, only comments of this type are displayed as annotations
- if no `doc:` comment is found in the entire DTD, all comments are considered annotations and displayed as such

When the option is disabled, all comments, regardless of their type, are considered annotations and displayed as such.

- **Use all Relax NG annotations as documentation** - When this option is selected, any element outside the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0`, is considered annotation and is displayed in the annotation window next to the **Content Completion Assistant** window and in the **Model** view. When this option is not selected, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

### XSL Preferences

XSL stylesheets are often used to create output in XHTML or XSL-FO. In addition to suggesting content completion options for XSLT stylesheet elements, Oxygen XML Editor can suggest elements from these vocabularies. To configure the XSL content completion options, [open the Preferences dialog box](#) and go to **Editor > Content Completion > XSL**.

The following options are available:

- **Automatically detect XHTML transitional or Formatting objects** - Detects if the output being generated is XHTML or FO and provides content completion for those vocabularies. Oxygen XML Editor analyzes the namespaces declared in the root element to find an appropriate schema.

If the detection fails, Oxygen XML Editor uses one of the following options:

- **None** - The **Content Completion Assistant** suggests only XSLT elements.
- **XHTML transitional** - The **Content Completion Assistant** includes XHTML Transitional elements as substitutes for `xsl:element`.
- **Formatting objects** - The **Content Completion Assistant** includes Formatting Objects (XSL-FO) elements as substitutes for `xsl:element`.
- **Custom schema** - If you want content completion hints for a different output vocabulary, enter the path to the schema for that vocabulary here. Supported schema types are DTD, XML Schema, RNG schema, or NVDL schema for inserting elements from the target language of the stylesheet.

You can choose an additional schema that will be used for documenting XSL stylesheets. Either select the built-in schema or choose a custom one. Supported schema types are XSD, RNG, RNC, DTD, and NDVL.

### **XPath Preferences**

Oxygen XML Editor provides content-completion support for XPath expressions. To configure the options for the content completion in XPath expressions, [open the Preferences dialog box](#) and go to **Editor > Content Completion > XPath**.

The following options are available:

- **Enable content completion for XPath expressions** - Enables [the Content Completion Assistant in XPath expressions](#) that you enter in the `match`, `select`, and `test` XSL attributes and also in the XPath toolbar.
  - **Include XPath functions** - When this option is selected, XPath functions are included in the content completion suggestions.
  - **Include XSLT functions** - When this option is selected, XSLT functions are included in the content completion suggestions.
  - **Include axes** - When this option is selected, XSLT axes are included in the content completion suggestions.
- **Show signatures of XSLT / XPath functions** - Makes the editor indicate the signature of the XPath function located at the caret position in a tooltip. See the [XPath Tooltip Helper](#) section for more information.
- **Function signature window background** - Specifies the background color of the tooltip window.
- **Function signature window foreground** - Specifies the foreground color of the tooltip window.

### **XSD Preferences**

Oxygen XML Editor provides content completion assistance when you are writing an XML Schema (XSD). To configure XSD preferences, [open the Preferences dialog box](#) and go to **Editor > Content Completion > XSD**. The options in this preferences page define what elements are suggested by the **Content Completion Assistant**, in addition to the ones from the XML Schema (defined by the `xs:annotation/xs:appinfo` elements).

The following options are available:

- **None** - The **Content Completion Assistant** offers only the XML Schema schema information.
- **ISO Schematron** - The **Content Completion Assistant** includes ISO Schematron elements in `xs:appinfo`.
- **Schematron 1.5** - The **Content Completion Assistant** includes Schematron 1.5 elements in `xs:appinfo`.
- **Other** - The **Content Completion Assistant** includes in `xs:appinfo` elements from an XML Schema identified by an URL.

### **JavaScript Preferences**

Oxygen XML Editor can provide content completion suggestions when you are writing JavaScript files. To configure content completion support for JavaScript, [open the Preferences dialog box](#) and go to **Editor > Content Completion > JavaScript**. You can configure the following options:

- **Enable content completion** - Enables the content completion support for JavaScript files.

- **Use built-in libraries** - Allows Oxygen XML Editor to include components (object names, properties, functions, and variables) collected from the built-in JavaScript library files when making suggestions.
- **Use defined libraries** - Oxygen XML Editor can also use JavaScript libraries to when making suggestions. List the paths (URIs) of any JavaScript files you want Oxygen XML Editor to use when making suggestions.



**Note:** The paths can contain editor variables such as \${pdu}, or \${oxygenHome}. You can make these paths relative to the project directory or installation directory.

## Colors Preferences

Oxygen XML Editor supports syntax highlighting of XML in the *Text mode* editor, DTD, Relax NG (XML and Compact Syntax), Java, JavaScript / JSON, Ant, PHP, CSS, XQuery, C++, C, Perl, Properties, SQL, Shell and Batch documents.

To configure syntax highlighting, [open the Preferences dialog box](#) and go to **Editor > Colors**.

To set syntax colors for a language, expand the listing for that language in the top panel to show the list of syntax items for that language. Use the color and style selectors to change how each syntax item is displayed. The results of your changes are shown in the preview panel. If you do not know the name of the syntax token that you want to configure, click on that token in the **Preview** area to select it.



**Note:** All default color sets come with a high-contrast variant, which is automatically used when you switch to a black-background or white-background high-contrast theme in your Windows operating system settings. The high-contrast theme will not overwrite any default color you set in **Colors** preferences page.

The settings for XML documents are used also in XSD, XSL, RNG documents. The **Preview** area has separate tabs for XML, XSD, XSL, RNG.

The **Enable nested syntax highlight** option controls if different content types mixed in the same file (like PHP, JS and CSS scripts inside an HTML file) are highlighted according with the color schemes defined for each content type.

## Elements / Attributes by Prefix Preferences

Oxygen XML Editor lets you specify different colors for XML elements and attributes with specific namespace prefixes.

To configure the **Elements / Attributes by Prefix** preferences, [open the Preferences dialog box](#) and go to **Editor > Colors > Elements / Attributes by Prefix**.

To change the syntax coloring for a specific namespace prefix, choose the prefix from the list, or add a new one using the **New** button, and use the color and style selectors to set the syntax highlighting style for that namespace prefix.



**Note:** Syntax highlighting is based on the literal namespace prefix, not the namespace that the prefix is bound to in the document.

If you want only the prefix, and not the whole element or attribute name, to be styled differently, select **Draw only the prefix with a separate color**.

## Open / Save Preferences

Oxygen XML Editor lets you control how files are opened and saved. To configure the **Open / Save** options, [open the Preferences dialog box](#) and go to **Editor > Open / Save**.

### Open

The following options apply to opening files:

- **Lock local resources** - When this option is enabled and you open a file from the local file system or a shared network drive, Oxygen XML Editor locks the file for the current user and the file cannot be modified by other users while the lock exists. However, other users are offered the possibility to edit the file, but without overwriting it. If they decide to continue, they will be asked to save the changes in a different file or drop them. Newly created files are *locked* when you first save them. If you enable this option with files already opened in Oxygen XML Editor, it will *lock* all the currently opened files. If you disable this option with files already opened, it will unlock them by deleting the corresponding .lock files.

- **Support for Special Characters** section - You can choose how you want Oxygen XML Editor to handle bidirectional text, Asian languages, or other special characters when they are detected. You can choose one of the following:
  - **Enable support for special characters**
  - **Disable support for special characters**
  - **Prompt for each document**
- **Disable bidirectional text support for documents larger than (Characters)** - Enabling bidirectional text editing support can affect performance on large files. When this option is selected, bidirectional editing is disabled for files exceeding the specified size, even if the **Enable support for special characters** option is selected.

## Save

The following options apply to saving files:

- **Safe save (only for local files)** - In the unlikely event of a failure of the **Save** action, this option provides increased protection from corruption of the original file. When this option is enabled, it saves the content to a temporary file and if the save is unsuccessful, the editor preserves its unsaved state status.
- **Make backup copy on save (only for local files)** - If selected, a backup copy is made when saving the edited document. This option is available only for local files (files that are stored on the local file system). The default backup file extension is `.bak`, but that can be changed.
- **Enable automatic save** - When selected, your documents are saved automatically, after a preset time interval.
- **Automatic save interval (minutes)** - Selects the interval, in minutes, for the automatic save action.
- **Check errors on save** - If enabled, Oxygen XML Editor runs a validation that checks your document for errors before saving it.
- **Save all files before transformation or validation** - Saves all open files before validating or transforming an XML document. This ensures that any dependencies are resolved when modifying the XML document and its XML Schema.
- **Save all files before calling external tools** - If selected, all files are saved before executing an [external tool](#).

## Performance

The following options cover performance issues when dealing with large files:

- **Optimize loading in the Text edit mode for files over (MB)** - File loading is optimized for reduced memory usage for any file with a size larger than this value. This is useful for editing large files, but it comes with [several restrictions](#) on memory-intensive operations.
- **Show warning when loading large documents** - Oxygen XML Editor will warn you if you open a file that is bigger than the specified size.
- **Optimize loading for documents with lines longer than (Characters)** - Line wrap is turned on for a document that contains lines that exceed the specified length. For a list of the restrictions applied to a document with long lines, see [the Editing Documents with Long Lines section](#).
- **Show warning when loading documents with long lines** - When selected, Oxygen XML Editor will warn you when you open a file with lines longer than the specified length. To reduce the length of lines in a file, [format and indent the document](#) after it is opened in the editor panel. For a list of the restrictions applied to a document with long lines, see [Editing Documents with Long Lines](#) on page 431.
- **Clear undo buffer on save** - If selected, Oxygen XML Editor clears its undo buffer when you save a document. Therefore, modifications made prior to saving the document cannot be undone. Select this option if you frequently encounter *out of memory* errors when editing large documents.
- **Consider application bundles to be directories when browsing** - This option is available only on the Mac OS X platform. When selected, the file browser dialog allows browsing inside an application bundle, as in a regular folder. Otherwise, the file browser dialog does not allow browsing inside an application bundle, as the Finder application does on Mac OS X.



**Note:** The same effect can be obtained by setting the property `apple.awt.use-file-dialog-packages` to `true` or `false` in the `Info.plist` descriptor file of the Oxygen XML Editor application:

```
<key>apple.awt.use-file-dialog-packages</key>
<string>false</string>
```

## Save Hooks Preferences

Oxygen XML Editor includes an option for automatically compiling LESS stylesheets. To set this option, [open the Preferences dialog box](#) and go to **Editor > Open / Save > Save hooks**.

The following option can be enabled or disabled:

- **Automatically compile LESS to CSS when saving** - If enabled, when you save a LESS file it will automatically be compiled to CSS (disabled by default).
  - ⚠ **Important:** If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

## Templates Preferences

This page groups the preferences for code templates and document templates:

- [Code Templates](#)
- [Document Templates](#)

### Code Templates Preferences

[Code templates](#) are code fragments that can be inserted at the current editing position. Oxygen XML Editor comes with a set of built-in templates for CSS, LESS, Schematron, XSL, XQuery, and XML Schema document types. You can also define your own code templates and share them with your colleagues using the template export and import functions.

To configure **Code Templates**, [open the Preferences dialog box](#) and go to **Editor > Templates > Code Templates**.

This preferences page contains a list of all the available code templates (both built-in and custom created ones) and a code preview area. You can disable any code template by deselecting it.

The following actions are available:

#### New

Opens the **Code template** dialog that allows you to define a new code template. You can define the following fields:

- **Name** - The name of the code template.
- **Description** - The description of the code template. that will appear in the **Code Templates** preferences page and in the tooltip message when selecting it from the **Content Completion Assistant**.
- **Associate with** - You can choose to set the code template to be associated with a specific type of editor or for all editor types.
- **Shortcut key** - Allows you to configure a shortcut key that can be used to insert the code template. The + character separates keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut is platform-independent and the following modifiers are used:
  - M1 represents the **Command** key on Mac OS X, and the **Ctrl** key on other platforms.
  - M2 represents the **Shift** key.
  - M3 represents the **Option** key on Mac OS X, and the **Alt** key on other platforms.
  - M4 represents the **Ctrl** key on Mac OS X, and is undefined on other platforms.
- **Content** - Text box where you define the content that is used when the code template is inserted.

#### Edit

Opens the **Code template** dialog and allows you to edit an existing code template. You can edit the following fields:

- **Description** - The description of the code template. that will appear in the **Code Templates** preferences page and in the tooltip message when selecting it from the **Content Completion Assistant**.
- **Shortcut key** - Allows you to configure a shortcut key that can be used to insert the code template. The + character separates keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut is platform-independent and the following modifiers are used:
  - M1 represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
  - M2 represents the **Shift** key.
  - M3 represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
  - M4 represents the **Ctrl** key on MacOS X, and is undefined on other platforms.
- **Content** - Text box where you define the content that is used when the code template is inserted.

### Duplicate

Creates a duplicate of the currently selected code template.

### Delete

Deletes the currently selected code template. This action is disabled for the built-in code templates.

### Export

Exports a file with code templates.

### Import

Imports a file with code templates that was created by the **Export** action.

You can use the following *editor variables* when you define a code template in the **Content** text box:

- `${caret}`  - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `${selection}`  - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}`  - To prompt for values at runtime, use the `ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')` editor variable. You can set the following parameters:
  - 'message' - The displayed message. Note the quotes that enclose the message.
  - type - Optional parameter, with one of the following values:

Parameter	
url	<p><b>Format:</b> <code> \${ask('message', url, 'default_value')} </code></p> <p><b>Description:</b> Input is considered a URL. Oxygen XML Editor checks that the provided URL is valid.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Input URL', url)} </code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL.</li> <li>• <code> \${ask('Input URL', url, 'http://www.example.com')} </code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL. The input field displays the default value <code>http://www.example.com</code>.</li> </ul>
password	<p><b>Format:</b> <code> \${ask('message', password, 'default')} </code></p> <p><b>Description:</b> The input is hidden with bullet characters.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Input password', password)} </code> - The displayed dialog box has the name <code>'Input password'</code> and the input is hidden with bullet symbols.</li> </ul>

Parameter	
	<ul style="list-style-type: none"> <li>• <code> \${ask('Input password', password, 'abcd') } </code> - The displayed dialog box has the name 'Input password' and the input hidden with bullet symbols. The input field already contains the default abcd value.</li> </ul>
generic	<p><b>Format:</b> <code> \${ask('message', generic, 'default')} </code></p> <p><b>Description:</b> The input is considered to be generic text that requires no special handling.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Hello world!')} </code> - The dialog box has a Hello world! message displayed.</li> <li>• <code> \${ask('Hello world!', generic, 'Hello again!')} </code> - The dialog box has a Hello world! message displayed and the value displayed in the input box is 'Hello again!'.</li> </ul>
relative_url	<p><b>Format:</b> <code> \${ask('message', relative_url, 'default')} </code></p> <p><b>Description:</b> Input is considered a URL. Oxygen XML Editor tries to make the URL relative to that of the document you are editing.</p> <p> <b>Note:</b> If the \$ask editor variable is expanded in content that is not yet saved (such as an <i>untitled</i> file, whose path cannot be determined), then Oxygen XML Editor will transform it into an absolute URL.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('File location', relative_url, 'C:/example.txt')} </code> - The dialog box has the name 'File location'. The URL inserted in the input box is made relative to the current edited document location.</li> </ul>
combobox	<p><b>Format:</b> <code> \${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')} </code></p> <p><b>Description:</b> Displays a dialog box that offers a drop-down list. The drop-down list is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated value (<code>real_value</code>).</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} </code> - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems. The associated value will be returned based upon your selection.</li> </ul> <p> <b>Note:</b> In this example Mac OS X is the default selected value and if selected it would return osx for the output.</p>
editable_combobox	<p><b>Format:</b> <code> \${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')} </code></p> <p><b>Description:</b> Displays a dialog box that offers a drop-down list with editable elements. The drop-down list is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated real value (<code>real_value</code>) or the value inserted when you edit a list entry.</p>

Parameter	
	<p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} </code> - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.</li> </ul>
radio	<p><b>Format:</b> <code> \${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')} </code></p> <p><b>Description:</b> Displays a dialog box that offers a series of radio buttons. Each radio button displays a 'rendered_value' and will return an associated real_value.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} </code> - The dialog box has the name 'Operating System'. The radio button group allows you to choose between the three operating systems.</li> </ul> <p> <b>Note:</b> In this example Mac OS X is the default selected value and if selected it would return osx for the output.</p>

- 'default-value' - optional parameter. Provides a default value.
  - `${timeStamp}`  - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
  - `${uuid}`  - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java **UUID** class.
  - `${id}`  - Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
  - `${cfn}`  - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
  - `${cfne}`  - Current file name with extension. The current file is the one currently opened and selected.
  - `${cf}`  - Current file as file path, that is the absolute file path of the current edited document.
  - `${cfld}`  - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
  - `${frameworksDir}`  - The path (as file path) of the [ OXYGEN\_DIR ] / frameworks directory.
  - `${pd}`  - Current project folder as file path. Usually the current folder selected in the Project View.
  - `${oxygenInstallDir}`  - Oxygen XML Editor installation folder as file path.
  - `${homeDir}`  - The path (as file path) of the user home folder.
  - `${pn}`  - Current project name.
  - `${env(VAR_NAME)}`  - Value of the VAR\_NAME environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the  `${system(var.name)}`  editor variable.
  - `${system(var.name)}`  - Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as -Dvar.name=var.value. If you are looking for operating system environment variables, use the  `${env(VAR_NAME)}`  editor variable instead.
  - `${date(pattern)}`  - Current date. The allowed patterns are equivalent to the ones in the [Java SimpleDateFormat class](#).
- Example:** YYYY-MM-dd;



**Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to <http://www.w3.org/TR/xmlschema-2/#date>. For details about xs:datetime, go to <http://www.w3.org/TR/xmlschema-2/#dateTime>.

## Document Templates Preferences

Oxygen XML Editor provides a selection of document templates that make it easier to create new documents in a variety of formats. The list of available templates is presented when you create a new document. You can add your own templates to this list by creating template files in a directory and adding that directory to the list of template directories that Oxygen XML Editor uses. To add a template directory, [open the Preferences dialog box](#) and go to **Editor > Templates > Document Templates**.

You can add new document template location folders and manage existing ones. You can also alter the order in which Oxygen XML Editor looks into these directories by using the **Up** and **Down** buttons.

## Spell Check Preferences

Oxygen XML Editor provides spell check support in the *text* and *author* edit modes. To configure the **Spell Check** options, [open the Preferences dialog box](#) and go to **Editor > Spell Check**.

The following options are available:

- **Spell checking engine** - Oxygen XML Editor ships with two spell check engines, *Hunspell* and *Java spell checker*. The two engines come with different dictionaries. When you select an engine here, the list of languages in the **Default language** option changes based on the available dictionaries for the engine you have chosen.
- **Automatic Spell Check** - When selected, Oxygen XML Editor checks spelling as you type and highlights misspelled words in the document.
  - **Select editors** - You can select which editors (and therefore which file types) will be automatically spelled checked. File types for which automatic spell check is generally not useful, like CSS and DTD, are excluded by default.
  - **Spell check highlight color** - Use this option to set the color used by the spell check engine to highlight spelling errors.

## Language Options Section

- **Default language** - The default language list allows you to choose the language used by the spell check engine when the language is not specified in the source file. You can [add additional dictionaries to the spell check engines](#).
- **Use "lang" and "xml:lang" attributes** - When this option is selected, the contents of an element with one of the lang or xml:lang attributes is checked in that language. When this option is enabled, choose between the following two options for instances **when these attributes are missing**:
  - **Use the default language** - If the lang and xml:lang attributes are missing, the selection in the **Default language** list is used.
  - **Do not check** - If the lang and xml:lang attributes are missing, the element is not checked.

## XML Spell Checking in Section

You can choose to check the spelling inside the following XML items:

- **Comments**
- **Attribute values**
- **Text**
- **CDATA**

## Options Section

- **Check capitalization** - When selected, the spell checker reports capitalization errors, for example a word that starts with lowercase after *etc.* or *i.e.*.

- **Check punctuation** - When selected, the spell checker checks punctuation. Misplaced white space and unusual sequences, like a period following a comma, are highlighted as errors.
- **Ignore mixed case words** - When selected, the spell checker does not check words containing mixed case characters (for example, *SpellChecker*).
- **Ignore acronyms** - When selected, acronyms are not reported as errors.
- **Ignore words with digits** - When selected, the spell checker does not check words containing digits (for example, *b2b*).
- **Ignore duplicates** - When selected, the spell checker does not signal two successive identical words as an error.
- **Ignore URL** - When selected, the spell checker ignores words looking like URLs or file names (for example, *www.oxygenxml.com* or *c:\boot.ini*).
- **Allow compounds words** - When selected, all words formed by concatenating two legal words with a hyphen (hyphenated compounds) are accepted. If recognized by the language, two words concatenated without hyphen (closed compounds) are also accepted.
- **Allow general prefixes** - When selected, a word formed by concatenating a recognized prefix and a legal word is accepted. For example if *mini-* is a registered prefix, the spell check engine accepts the word *mini-computer*.
- **Allow file extensions** - When selected, the spell checker accepts any word ending with recognized file extensions (for example, *myfile.txt* or *index.html*).

## Ignore Elements Section

You can use the **Add** and **Remove** buttons to configure a list of element names or XPath expressions to be ignored by the spell checker. The following restricted set of XPath expressions are supported:

- '/' and '//' separators
- '\*' wildcard

An example of an allowed XPath expression is: */a/\*b*.

## Spell Check Dictionaries Preferences

To set the Dictionaries preferences, [open the Preferences dialog box](#) and go to **Editor > Spell Check > Dictionaries**. This page allows you to configure the dictionaries and term lists (.tdi files) that Oxygen XML Editor uses and to choose where to save new learned words.

The following options are valid when Oxygen XML Editor uses the Hunspell spell checking engine:

- **Dictionaries and term lists default folder** - Displays the default location where the dictionaries and term lists that Oxygen XML Editor uses are stored.
- **Include dictionaries and term list from** - Specifies a location where you can store dictionaries and term lists that are different from the default one.



**Note:** The spell checker takes into account dictionaries and term lists collected both from the default and custom locations and multiple dictionaries and term lists from the same language are merged into generic ones (for example, *en\_UK.dic* from the default location is merged with *en\_US.dic* from a custom location, and the result is that a third file is created for a generic dictionary called *en.dic*). However, if there is already a generic dictionary (for example, *en.dic*) saved in either the default or custom location, the other specific dictionaries (for example, *en\_UK.dic* and *en\_US.dic*) will not be merged and the existing generic dictionary will simply be used. Also, if the additional location contains a file with the same name as one from the default location, the file in the additional location takes precedence over the file from the default location.

- **Save learned words in the following location** - Specifies the target where the newly learned words are saved. By default, the target is the application preferences folder, but you can also choose a custom location.
- **Delete learned words** - Opens the list of learned words, allowing you to select the items you want to remove, without deleting the dictionaries and term lists.

## Document Checking Preferences

To configure the **Document Checking** options, [open the Preferences dialog box](#) and go to **Editor > Document Checking**. This preferences page contains preferences for configuring how a document is checked for both well-formedness errors and validation errors.

The following options are available:

- **Maximum number of validation highlights** - If validation generates more errors than the number from this option only the first errors up to this number are highlighted in editor panel and on stripe that is displayed at right side of editor panel. This option is applied both for [automatic validation](#) and [manual validation](#).
- **Validation error highlight color** - The color used to highlight validation errors in the document.
- **Validation warning highlight color** - The color used to highlight validation warnings in the document.
- **Validation success color** - The color used to highlight in the vertical ruler bar the success of the validation operation.
- **Always show validation status** - If this option is selected the line at the bottom of the editor panel which presents the current validation error or warning is always visible. This is useful to avoid scrolling problems when **Automatic validation** is enabled and the vertical scroll bar may change position due to displaying an error message while the document is edited.
- **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen XML Editor.
- **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

At the bottom of the preferences page you can [choose whether or not the saved options will be shared with other users by selecting Global or Project storage options](#).

## Mark Occurrences Preferences

To configure the **Mark Occurrences** options, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**:

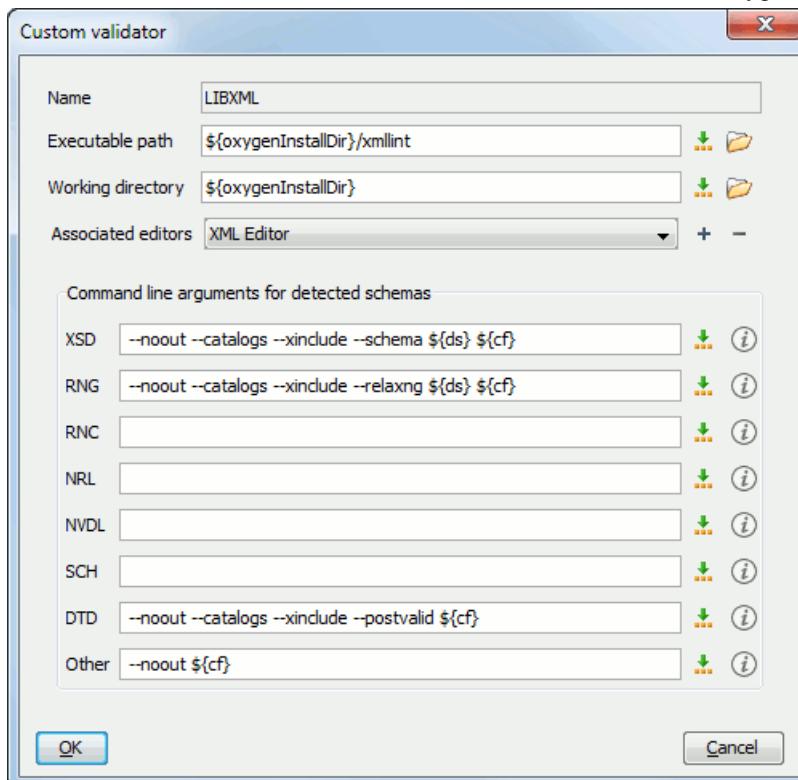
The following preferences are available in this preferences page:

- **XML files** - Activates the [Highlight IDs Occurrences](#) feature in XML files.
- **XSLT files** - Activates the [Highlight Component Occurrences](#) feature in XSLT files.
- **XML Schema files and WSDL files** - Activates the [Highlight Component Occurrences](#) feature in XSD and WSDL files.
- **RNG files** - Activates the highlight component occurrences feature in RNG files.
- **Schematron files** - Activates the [Highlight Component Occurrences](#) feature in Schematron files.
- **Ant files** - Activates the [Highlight Component Occurrences](#) feature in Ant files.
- **Declaration highlight color** - Color used to highlight the component declaration.
- **Reference highlight color** - Color used to highlight component references.

## Custom Validation Engines Preferences

To configure the options for *Custom Validation Engines*, [open the Preferences dialog box](#) and go to **Editor > Custom Validations**.

If you want to add a new custom validation tool or edit the properties of an exiting one you can use the **Custom Validator** dialog displayed by pressing the **New** button or the **Edit** button.



**Figure 413: Edit a Custom Validator**

The configurable parameters of a custom validator are as follows:

- **Name** - Name of the custom validation engine that will be displayed in the Validation toolbar drop-down list.
- **Executable path** - Path to the executable file of the custom validation tool. You can insert here *editor variables* like \${home}, \${pd}, \${oxygenInstallDir}, etc.
- **Working directory** - The working directory of the custom validation tool.
- **Associated editors** - The editors which can perform validation with the external tool: the XML editor, the XSL editor, the XSD editor, etc.
- **Command line arguments for detected schemas** - Command line arguments used in the commands that validate the current edited file against different types of schema: W3C XML Schema, Relax NG full syntax, Relax NG compact syntax, NVDL, Schematron, DTD, etc.. The arguments can include any custom switch (like -rng) and the following editor variables:
  - \${cf} - Current file as file path, that is the absolute file path of the current edited document.
  - \${currentFileURL} - Current file as URL, that is the absolute file path of the current edited document represented as URL.
  - \${ds} - The path of the detected schema as a local file path for the current validated XML document.
  - \${dsu} - The path of the detected schema as an URL for the current validated XML document.

### Increasing the stack size for validation engines

To prevent the appearance of a *StackOverflowException*, use one of the following methods:

- use the **com.oxygenxml.stack.size.validation.threads** property to increase the size of the stack for validation engines. The value of this property is specified in bytes. For example, to set a value of one megabyte specify 1x1024x1024=1048576;
- increase the value of the **-Xss** parameter.



**Note:** Increasing the value of the **-Xss** parameter affects all the threads of the application.

## CSS Validator Preferences

To configure the **CSS Validator** preferences, [open the Preferences dialog box](#) and go to **CSS Validator**.

You can configure the following options for the built-in **CSS Validator** of Oxygen XML Editor:

- **Profile** - Selects one of the available validation profiles: **CSS 1**, **CSS 2**, **CSS 2.1**, **CSS 3**, **CSS 3 with Oxygen extensions**, **SVG**, **SVG Basic**, **SVG Tiny**, **Mobile**, **TV Profile**, **ATSC TV Profile**. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties plus the *CSS extensions specific for Oxygen* that can be used in *Author mode*. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.
- **Media type** - Selects one of the available mediums: **all**, **aural**, **braille**, **embossed**, **handheld**, **print**, **projection**, **screen**, **tty**, **tv**, **presentation**, **oxygen**
- **Warning level** - Sets the minimum severity level for reported validation warnings. Can be one of: **All**, **Normal**, **Most Important**, **No Warnings**
- **Ignore properties** - You can type comma separated patterns that match the names of CSS properties that will be ignored at validation. As wildcards you can use:
  - \* to match any string
  - ? to match any character
- **Recognize browser CSS extensions (applies also to content completion)** - If checked, Oxygen XML Editor recognizes (no validation is performed) browser-specific CSS properties. The **Content Completion Assistant** lists these properties at the end of its list, prefixed with the following particles:
  - -moz- for Mozilla
  - -ms- for Internet Explorer
  - -o- for Opera
  - -webkit- for Safari/WebKit

## XML Preferences

This section describes the panels that contain the user preferences related with XML.

### XML Catalog Preferences

To configure the **XML Catalog** options, [open the Preferences dialog box](#) and go to **XML > XML Catalog**.

The following options are available:

- **Prefer** - the prefer setting determines whether public identifiers specified in the catalog are used in favor of system identifiers supplied in the document. Suppose you have an entity in your document for which both a public identifier and a system identifier has been specified, and the catalog only contains a mapping for the public identifier (e.g., a matching public catalog entry). If **public** is selected, the URI supplied in the matching public catalog entry is used. If **system** is selected, the system identifier in the document is used.



**Note:** If the catalog contained a matching system catalog entry giving a mapping for the system identifier, that mapping would have been used, the public identifier would never have been considered, and the setting of override would have been irrelevant.

Generally, the purpose of catalogs is to override the system identifiers in XML documents, so **Prefer** should usually be **public** in your catalogs.

- When using catalogs it is sometimes useful to see what catalog files are parsed, if they are valid or not, and what identifiers are resolved by the catalogs. The **Verbosity** option selects the detail level of such logging messages of the XML catalog resolver that will be displayed in the **Catalogs** view at the bottom of the window:
  - **None** - No message is displayed by the catalog resolver when it tries to resolve a URI reference, a SYSTEM one or a PUBLIC one with the XML catalogs specified in this panel.
  - **Unresolved entities** - Only the logging messages that track the failed attempts to resolve references are displayed.
  - **All messages** - The messages of both failed attempts and successful ones are displayed.

- If **Resolve schema locations also through system mappings** is enabled, Oxygen XML Editor analyzes both *uri* and *system* mappings in order to resolve the location of schema.
- If **Process namespaces through URI mappings for XML Schema** is selected then the target namespace of the imported XML Schemas is resolved through the *uri* mappings. The namespace is taken into account only when the schema specified in the *schemaLocation* attribute was not resolved successfully.
- If the **Use default catalog** option is checked the first XML catalog which Oxygen XML Editor will use to resolve references at document validation and transformation will be a default built-in catalog. This catalog maps such references to the built-in local copies of the schemas of the Oxygen XML Editor frameworks: DocBook, DITA, TEI, XHTML, SVG, etc.

You can also add or configure catalogs at framework level in the [Document Type Association](#) preferences page.

When you add, delete or edit an XML catalog to / from the list, reopen the currently edited files which use the modified catalog or run [the Validate action](#) so that the XML catalog changes take full effect.

## XML Parser Preferences

To configure the **XML Parser** options, [open the Preferences dialog box](#) and go to **XML > XML Parser**.

The configurable options of the built-in XML parser are the following:

- **Enable parser caching (validation and content completion)** - enables re-use of internal models when validating and provides content completion in opened XML files which reference the same schemas (grammars) like DTD, XML Schema, RelaxNG.
- **Ignore the DTD for validation if a schema is specified** - forces validation against a referenced schema (W3C XML Schema, Relax NG schema) even if the document includes also a DTD DOCTYPE declaration. This option is useful when the DTD declaration is used only to declare DTD entities and the schema reference is used for validation against a W3C XML Schema or a Relax NG schema.



**Note:** Schematron schemas are treated as additional schemas. The validation of a document associated with a DTD and referencing a Schematron schema is executed against both the DTD and the Schematron schema, regardless of the value of the **Ignore the DTD for validation if a schema is specified** option.

- **Enable XInclude processing** - enables XInclude processing. If checked, the XInclude support in Oxygen XML Editor is turned on for validation, rendering in Author mode and transformation of XML documents.
- **Base URI fix-up** - according to the specification for XInclude, processors must add an `xml:base` attribute to elements included from locations with a different base URI. Without these attributes, the resulting infoset information would be incorrect.

Unfortunately, these attributes make XInclude processing not transparent to Schema validation. One solution to this is to modify your schema to allow `xml:base` attributes to appear on elements that might be included from different base URIs.

If the addition of `xml:base` and / or `xml:lang` is undesired by your application, you can disable base URI fix-up.

- **Language fix-up** - the processor will preserve language information on a top-level included element by adding an `xml:lang` attribute if its include parent has a different [language] property. If the addition of `xml:lang` is undesired by your application, you can disable the language fix-up.
- **DTD post-validation** - enable this option to validate an XML file against the associated DTD, after all the content merged to the current XML file using XInclude was resolved. In case you disable this option, the current XML file is validated against the associated DTD before all the content merged to the current XML file using XInclude is resolved.

## XML Schema Preferences

To configure the **XML Schema** options, [open the Preferences dialog box](#) and go to **XML > XML Parser > XML Schema**.

This preferences page allows you to configure the following options:

- **Default XML Schema version** - Allows you to select the version of W3C XML Schema: XML Schema **1.0** or XML Schema **1.1**.
 

 **Note:** You are also able to set the XML Schema version using the **Customize** option in [New dialog box](#).
- **Default XML Schema validation engine** - Allows you to set the default XML Schema validation engine either to **Xerces** or **Saxon EE**.

### Xerces validation features

- **Enable full schema constraint checking** (<http://apache.org/xml/features/validation/schema-full-checking>) - Sets the *schema-full-checking* feature to true. This enables a validation of the parsed XML document against a schema (W3C XML Schema or DTD) while the document is parsed.
- **Enable honour all schema location feature** (<http://apache.org/xml/features/honour-all-schema-location>) - Sets the *honour-all-schema-location* feature to true. All the files that declare W3C XML Schema components from the same namespace are used to compose the validation model. In case this option is disabled, only the first W3C XML Schema file that is encountered in the XML Schema import tree is taken into account.
- **Enable full XPath 2.0 in assertions and alternative types**  
(<http://apache.org/xml/features/validation/cta-full-xpath-checking>) - When enabled (default value), you can use the full XPath support in assertions and alternative types. Otherwise, only the XPath support offered by the Xerces engine is available.
- **Assertions can see comments and processing instructions**  
(<http://apache.org/xml/features/validation/assert-comments-and-pi-checking>) - Controls whether comments and processing instructions are visible to the XPath expression used for defining an assertion in XSD 1.1.

### Saxon validation features

- **Multiple schema imports** - Forces xs:import to fetch the referenced schema document. By default, the xs:import fetches the document only if no schema document for the given namespace has already been loaded. With this option in effect, the referenced schema document is loaded unless the absolute URI is the same as a schema document already loaded.
- **Assertions can see comments and processing instructions** - Controls whether comments and processing instructions are visible to the XPath expression used to define an assertion. By default (unlike Saxon 9.3), they are not made visible.

### Relax NG Preferences

To configure the **Relax NG** options, [open the Preferences dialog box](#) and go to **XML > XML Parser > Relax NG**.

The following options are available in this page:

- **Check feasibly valid** - Checks whether Relax NG documents can be transformed into valid documents by inserting any number of attributes and child elements anywhere in the tree.
 

 **Note:** Enabling this option disables the **Check ID/IDREF** option.
- **Check ID/IDREF** - Checks the ID/IDREF matches when a Relax NG document is validated.
- **Add default attribute values** - Default values are given to the attributes of documents validated using Relax NG. These values are defined in the Relax NG schema.

### Schematron Preferences

To configure the **Schematron** options, [open the Preferences dialog box](#) and go to **XML > XML Parser > Schematron**.

The following options are available in this preferences page:

- **Schematron XPath Version** - selects the version of XPath for the expressions that are allowed in Schematron assertion tests: 1.0 or 2.0. This option is applied both in standalone Schematron schemas and in embedded Schematron rules, both in Schematron 1.5 and in ISO Schematron.
- **Optimize (visit-no-attributes)** - in case your ISO Schematron assertion tests do not contain the attributes axis you should check this option for faster ISO Schematron validation.

- **Allow foreign elements (allow-foreign)** - enables support for `allow-foreign` on ISO Schematron. This option is used to pass non-Schematron elements to the generated stylesheet.
- **Use Saxon EE (schema aware) for xslt2 query binding** - when enabled, Saxon EE is used for `xslt2` query binding. In case this option is disabled, Saxon PE is used.
- **Enable Schematron Quick Fixes (SQF) support** - Allows you to enable or disable the support for quick fixes in Schematron files. This option is enabled by default.

## XML Instances Generator Preferences

To configure the **XML Instances Generator** options, [open the Preferences dialog box](#) and go to **XML > XML Instances Generator**. It sets the default parameters of the **Generate Sample XML Files** tool that is available on the **Tools** menu.

The options of the tool that generates XML instance documents based on a W3C XML Schema are the following:

- **Generate optional elements** - If checked, the elements declared optional in the schema will be generated in the XML instance.
- **Generate optional attributes** - If checked, the attributes declared optional in the schema will be generated in the XML instance.
- **Values of elements and attributes** - Specifies what values are generated in elements and attributes of the XML instance. It can have one of the values:
  - **None** - no values for the generated elements and attributes
  - **Default** - the value is the element name or attribute name
  - **Random** - a random value
- **Preferred number of repetitions** - If the values set here is greater than `maxOccurs`, then the `maxOccurs` is used.
- **Maximum recursivity level** - For recursive type definitions this parameter specifies the number of levels of recursive elements inserted in the parent element with the same name.
- **Type alternative strategy** - Specifies how the element type alternatives are generated in the XML instance:
  - **First** - the first element type alternative whose XPath condition is true is used;
  - **Random** - a random element type alternative whose XPath condition is true is used;

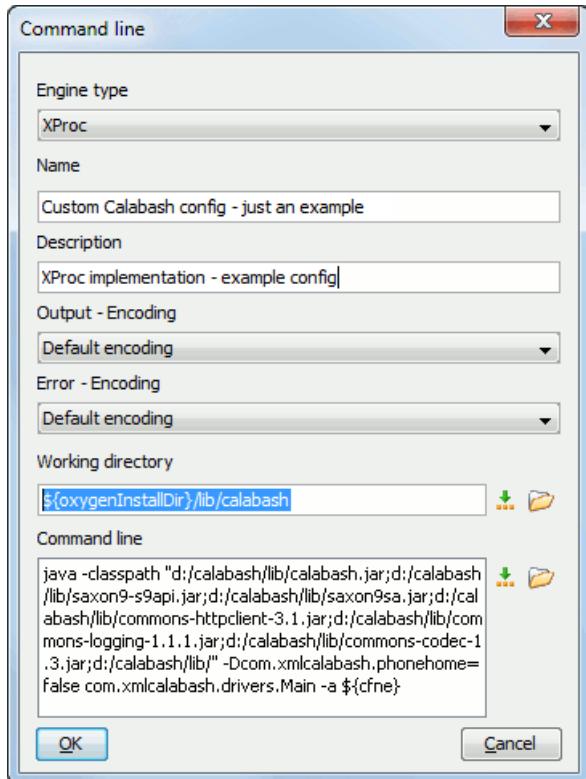
 **Note:** In case no XPath condition is true, the default element type alternative is used.
-  **Note:** To evaluate an XPath expression, either the values of the attributes defined in the **Options** tab of the **XML Instance Generator** dialog, or the attributes values defined in the XML Schema are used.
- **Choice strategy** - For choice element models specifies what choice will be generated in the XML instance:
  - **First** - the first choice is selected from the choice definition and an instance of that choice is generated in the XML instance document.
  - **Random** - a random choice is selected from the choice definition and an instance of that will be generated.
- **Generate the other options as comments** - If checked, the other options of the choice element model (the options which are not selected) will be generated inside an XML comment in the XML instance.
- **Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an `a` element the generated values are: `a1, a2, a3`, etc. If not checked, the value is the name of the type of that element / attribute, for example: `string, decimal`, etc.
- **Maximum length** - The maximum length of string values generated for elements and attributes.
- **Discard optional elements after nested level** - The optional elements that exceed the specified nested level are discarded. This option is useful for limiting deeply nested element definitions that can quickly result in very large XML documents.

## XProc Engines Preferences

Oxygen XML Editor comes with a built-in XProc engine called *Calabash*. An external XProc engine can be configured in this panel.

When **Show XProc messages** is selected all messages emitted by the XProc processor during a transformation will be presented in the results view.

For an external engine the value of the **Name** field will be displayed in the XProc transformation scenario and in the command line that will start it.



**Figure 414: Creating an XProc external engine**

Other parameters that can be set for an XProc external engine are the following: , and the error stream of the engine, the working directory of the command that will start the engine. The encodings will be used for reading and displaying the output of the engine. The working directory and

- a textual description that will appear as tooltip where the XProc engine will be used
- the encoding for the output stream of the XProc engine, used for reading and displaying the output messages
- the encoding for the error stream of the XProc engine, used for reading and displaying the messages from the error stream
- the working directory for resolving relative paths
- the command line that will run the XProc engine as an external process; the command line can use *built-in editor variables* and *custom editor variables* for parametrizing a file path

 **Note:** You can configure the Saxon processor using the `saxon.config` file. For further details about configuring this file go to <http://www.saxonica.com/documentation9.5/index.html#!configuration/configuration-file>.

 **Note:** You can configure Calabash using the `calabash.config` file.

 **Note:** These files are located in `[OXYGEN_DIR]\lib\xproc\calabash`. In case they do not exist, you have to create them.

## XSLT-FO-XQuery Preferences

To configure the **XSLT/FO/XQuery** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery**. This panel contains only the most generic options for working with XSLT / XSL-FO / XQuery processors. The more specific options are grouped in other panels linked as child nodes of this panel in the tree of the **Preferences** dialog.

There is only one generic option available:

**Create transformation temporary files in system temporary directory** - It should be selected only when the temporary files necessary for performing transformations are created in the same folder as the source of the transformation (the default behavior, when this option is not selected) and this breaks the transformation. An example of breaking the transformation is when the transformation processes all the files located in the same folder as the source of the transformation, which will include the temporary files, and the result is incorrect or the transformation fails due to this fact.

## XSLT Preferences

To configure the **XSLT** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT**.

Oxygen XML Editor gives you the possibility to use an XSLT transformer implemented in Java (other than the XSLT transformers that come bundled with Oxygen XML Editor). To use a different XSLT transformer, specify the name of the transformer factory class. Oxygen XML Editor sets this transformer factory class as the value of the Java property `javax.xml.transform.TransformerFactory`.

You can customize the following XSLT preferences:

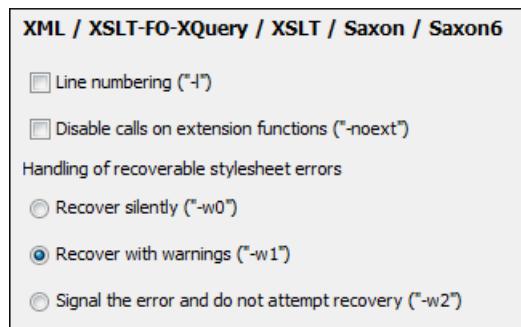
- **Value** - Allows the user to enter the name of the transformer factory Java class.
- **XSLT 1.0 Validate with** - allows you to set the XSLT engine used for validation of XSLT 1.0 documents.
- **XSLT 2.0 Validate with** - allows you to set the XSLT Engine used for validation of XSLT 2.0 documents.
- **XSLT 3.0 Validate with** - allows you to set the XSLT Engine used for validation of XSLT 3.0 documents.



**Note:** Saxon-HE does not implement any XSLT 3.0 features. Saxon-PE implements a selection of XSLT 3.0 (and XPath 3.0) features, with the exception of schema-awareness and streaming. Saxon-EE implements additional features relating to streaming (processing of a source document without constructing a tree in memory. For further details about XSLT 3.0 conformance, go to <http://www.saxonica.com/documentation/index.html#!conformance/xslt30>.

## Saxon6 Preferences

To configure the **Saxon 6** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon 6**.



**Figure 415: The Saxon 6 XSLT Preferences Panel**

The built-in Saxon 6 XSLT processor can be configured with the following options:

- **Line numbering** - Specifies whether line numbers are maintained and reported in error messages for the XML source document.

- **Disable calls on extension functions** - If enabled, external functions called is disallowed. Checking this is recommended in an environment where untrusted stylesheets may be executed. Also disables user-defined extension elements, together with the writing of multiple output files, all of which carry similar security risks.
- **Handling of recoverable stylesheet errors** - Allows the user to choose how dynamic errors are handled. Either one of the following options can be selected:
  - **recover silently** - Continue processing without reporting the error.
  - **recover with warnings** - Issue a warning but continue processing.
  - **signal the error and do not attempt recovery** - Issue an error and stop processing.

#### Saxon-HE/PE/EE Preferences

To configure the **Saxon HE/PE/EE** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE**.

Oxygen XML Editor allows you to configure the following XSLT options for the Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE):

- **Use a configuration file ("<-config>")** - Sets a Saxon 9.6.0.5 configuration file that is executed for XSLT transformation and validation processes.
- **Version warnings ("<-versmsg>")** - Warns you when the transformation is applied to an XSLT 1.0 stylesheet.
- **Line numbering ("<-l>")** - Error line numbers are included in the output messages.
- **Debugger trace into XPath expressions (applies to debugging sessions)** - Instructs the [XSLT Debugger](#) XSLT Debugger to step into XPath expressions.
- **Expand attributes defaults ("<-expand>")** - Specifies whether or not the attributes defined in the associated DTD or XML Schema are expanded in the output of the transformation you are executing.
- **DTD validation of the source ("<-dtd>")** - The following options are available:
  - **On** - Requests DTD validation of the source file and of any files read using the `document()` function.
  - **Off** - (default setting) Suppresses DTD validation.
  - **Recover** - Performs DTD validation but treats the errors as non-fatal.



**Note:** Any external DTD is likely to be read even if not used for validation, since DTDs can contain definitions of entities.

- **Recoverable errors ("<-warnings>")** - Policy for handling recoverable errors in the stylesheet: Allows you to choose how dynamic errors are handled. One of the following options can be selected:
  - **Recover silently ("silent")** - Continues processing without reporting the error.
  - **Recover with warnings ("recover")** - (default setting) Issues a warning but continues processing.
  - **Signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.
- **Strip whitespaces ("<-strip>")** - Strip whitespaces feature can be one of the following three options:
  - **All ("all")** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document.
  - **Ignorable ("ignorable")** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None ("none")** - default setting. No whitespaces are stripped before further processing. However, whitespace are stripped if this is specified in the stylesheet using `xsl:strip-space`.
- **Optimization level ("<-opt>")** - Set the optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization). This option allows optimization to be suppressed in cases where reducing the compiling time is important, where optimization conflicts with debugging, or causes extension functions with side-effects to behave unpredictably.

The advanced options available only in Saxon Professional Edition (PE) and Enterprise Edition (EE) are:

- **Allow calls on extension functions ("‐ext")** - If checked, the stylesheet is allowed to call external Java functions. This does not affect calls on integrated extension functions, including Saxon and EXSLT extension functions. This option is useful when loading an untrusted stylesheet, such as from a remote site using an `http://[URL]`. It ensures that the stylesheet cannot call arbitrary Java methods and thus gain privileged access to resources on your machine.
- **Register Saxon-CE extension functions and instructions** - Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.6.0.5 processors.



**Note:** Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Editor only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about [executing such a stylesheet on Saxonica's website](#).

The advanced options available only in Saxon Enterprise Edition (EE) are:

- **XML Schema version** - Use this option to change the default XML Schema version. To change the default XML Schema version, [open the Preferences dialog box](#) and go to **XML > XML Parser > XML Schema**.
  - Note:** This option is available when you configure the Saxon EE advanced options from a transformation scenario.
- **Validation of the source file ("‐val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and specifies that the source documents should be parsed with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("‐outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.
- **Generate bytecode ("‐generateByteCode:(on|off)")** - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to <http://www.saxonica.com/documentation9.5/index.html#!javadoc>.

#### Saxon HE/PE/EE Advanced Preferences

To configure the **Saxon HE/PE/EE Advanced** preferences, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE > Advanced**.

XML / XSLT-FO-XQuery / XSLT / Saxon / Saxon-HE/PE/EE / Advanced	
URI Resolver class name ("‐r")	<input type="text"/>
Collection URI Resolver class name ("‐cr")	<input type="text"/>
<b>The resolver classes must be present in the scenario extensions.</b>	

**Figure 416: The Saxon HE/PE/EE XSLT Advanced Preferences Panel**

You can configure the following advanced XSLT options for the Saxon 9.6.0.5 transformer (all three editions: Home Edition, Professional Edition, Enterprise Edition):

- **URI Resolver class name ("‐r")** - Specifies a custom implementation for the URI resolver used by the XSLT Saxon 9.6.0.5 transformer (the `-r` option when run from the command line). The class name must be fully specified and the corresponding `.jar` or `.class` extension must be configured from [the dialog for configuring the XSLT extension](#) for the particular transformation scenario.

- **Collection URI Resolver class name ("‐cr")** - Specifies a custom implementation for the Collection URI resolver used by the XSLT Saxon 9.6.0.5 transformer (the ‐cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XSLT extension](#) for the particular transformation scenario.

### XSLTProc Preferences

To configure **XSLTProc** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT > XSLTProc**.

The options of the XSLTProc processor are the same as the ones available in the command line:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when XSLTProc is used as transformer in [XSLT transformation scenarios](#).
- **Skip loading the document's DTD** - If checked, the DTD specified in the DOCTYPE declaration will not be loaded.
- **Do not apply default attributes from document's DTD** - If checked, the default attributes declared in the DTD and not specified in the document are not included in the transformed document.
- **Do not use Internet to fetch DTD's, entities or docs** - If checked, the remote references to DTD's and entities are not followed.
- **Maximum depth in templates stack** - If this limit of maximum templates depth is reached the transformation ends with an error.
- **Verbosity** - If checked, the transformation will output detailed status messages about the transformation process in the **Warnings** view.
- **Show version of libxml and libxslt used** - If checked, Oxygen XML Editor will display in the **Warnings** view the version of the **libxml** and **libxslt** libraries invoked by XSLTProc.
- **Show time information** - If checked, the **Warnings** view will display the time necessary for running the transformation.
- **Show debug information** - If checked, the **Warnings** view will display debug information about what templates are matched, parameter values, etc.
- **Show all documents loaded during processing** - If checked, Oxygen XML Editor will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen XML Editor will display in the **Warnings** view a table with all the matched templates, and for each template will display: the match XPath expression, the template name, the number of template modes, the number of calls, the execution time.
- **Show the list of registered extensions** - If checked, Oxygen XML Editor will display in the **Warnings** view a list with all the registered extension functions, extension elements and extension modules.
- **Refuses to write to any file or resource** - If checked, the XSLTProc processor will not write any part of the transformation result to an external file on disk. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.
- **Refuses to create directories** - If checked, the XSLTProc processor will not create any directory during the transformation process. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

### MSXML Preferences

To configure the MSXML options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT > MSXML**.

The options of the MSXML 3.0 and 4.0 processors are the same as [the ones available in the command line for the MSXML processors](#):

- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default, MSXSL instructs the parser to resolve external definitions such as document type definition (DTD), external subsets or external entity references when parsing the source and style sheet documents. If this option is checked the resolution is disabled.

- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.
- **Show time information** - If checked, the relative speed of various transformation steps can be measured:
  - The time to load, parse, and build the input document.
  - The time to load, parse, and build the stylesheet document.
  - The time to compile the stylesheet in preparation for the transformation.
  - The time to execute the stylesheet.
- **Start transformation in this mode** - Although stylesheet execution usually begins in the empty mode, this default may be changed by specifying another mode. Changing the start mode allows execution to jump directly to an alternate group of templates.

### *MSXML.NET Preferences*

To configure the **MSXML.NET** options, [open the \*Preferences\* dialog box](#) and go to **XML > XSLT/FO/XQuery > XSLT > MSXML.NET**.

The options of the MSXML.NET processor are:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when MSXML.NET is used as transformer in the [\*XSLT transformation scenario\*](#).
- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default MSXML.NET resolves external definitions such as DTD external subsets or external entity references when parsing source XML document and stylesheet document. Using this option you can disable this behaviour. Note, that it may affect also the validation process for the XML document.
- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.
- **Show time information** - If checked, the relative speed of various transformation steps can be measured:
  - The time to load, parse, and build the input document.
  - The time to load, parse, and build the stylesheet document.
  - The time to compile the stylesheet in preparation for the transformation.
  - The time to execute the stylesheet.
- **Forces ASCII output encoding** - There is a known problem with .NET 1.X XSLT processor (`System.Xml.Xsl.XsltTransform` class): it doesn't support escaping of characters as XML character references when they cannot be represented in the output encoding. That means that when you output a character that cannot be represented in output encoding, it will be outputted as '?'. Usually this happens when output encoding is set to ASCII. With this option checked the output is forced to be ASCII encoded and all non-ASCII characters get escaped as XML character references (&#nnnn; form).
- **Allow multiple output documents** - This option allows to create multiple result documents using [\*the `exsl:document` extension element\*](#).
- **Use named URI resolver class** - This option allows to specify a custom URI resolver class to resolve URI references in `xsl:import` and `xsl:include` instructions (during XSLT stylesheet loading phase) and in `document()` function (during XSL transformation phase).
- **Assembly file name for URI resolver class** - The previous option specifies partially or fully qualified URI resolver class name, e.g. `Acme.Resolvers.CacheResolver`. Such name requires additional assembly specification using this option or the next option, but fully qualified class name (which always includes an assembly specifier) is all-sufficient. See MSDN for more info about [\*fully qualified class names\*](#). This option specifies a file name of the assembly, where the specified resolver class can be found.
- **Assembly GAC name for URI resolver class** - This option specifies partially or fully qualified name of the assembly in the [\*global assembly cache\*](#) (GAC), where the specified resolver class can be found. See MSDN for more info about [\*partial assembly names\*](#). Also see the previous option.

- **List of extension object class names** - This option allows to specify *extension object* classes, whose public methods then can be used as extension functions in an XSLT stylesheet. It is a comma-separated list of namespace-qualified extension object class names. Each class name must be bound to a namespace URI using prefixes, similar to providing XSLT parameters.
- **Use specified EXSLT assembly** - MSXML.NET supports a rich library of the *EXSLT* and *EXSLT.NET extension functions* embedded or in a plugged in EXSLT.NET library. EXSLT support is enabled by default and cannot be disabled in this version. If you want to use an external EXSLT.NET implementation instead of a built-in one use this option.
- **Credential loading source xml** - This option allows to specify user credentials to be used when loading XML source documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).
- **Credential loading stylesheet** - This option allows to specify user credentials to be used when loading XSLT stylesheet documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).

## XQuery Preferences

To configure the **XQuery** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XQuery**.

The generic XQuery preferences are the following:

- **XQuery validate with** - Allows you to select the processor that validates XQuery documents. In case you are validating an XQuery file that has an associated validation scenario, Oxygen XML Editor uses the processor specified in the scenario. If no validation scenario is associated, but the file has an associated transformation scenario, the processor specified in the scenario is used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor.
  - **Size limit of Sequence view (MB)** - When the result of an XQuery transformation is [set in the transformation scenario as sequence](#) the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceed, go to the **Sequence** view and click **More results available** to extract more data from the database.
  - **Format transformer output** - Specifies whether the output of the transformer is formatted and indented (pretty printed).
-  **Note:** This option is ignored if you choose **Sequence** (lazy extract data from a database) from the associated transformation scenario.
- **Create structure indicating the type nodes** - If checked, Oxygen XML Editor takes the results of a query and creates an XML document containing copies of all items in the sequence, suitably wrapped.
-  **Note:** This option is ignored if you choose **Sequence** (lazy extract data from a database) from the associated transformation scenario.

## Saxon HE/PE/EE Preferences

To configure the **Saxon HE/PE/EE** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE**.

The XQuery preferences for the Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

- **Use a configuration file ("<-config>")** - Sets a Saxon 9 configuration file that is used for XQuery transformation and validation
- **Recoverable errors ("<-warnings>")** - Allows the user to choose how dynamic errors are handled. The following options can be selected:
  - **recover silently ("silent")** - Continues processing without reporting the error.
  - **recover with warnings ("recover")** - Issues a warning but continues processing.
  - **signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.
- **Strip whitespaces ("<-strip>")** - Can have one of the following values:

- **All ("all")** - Strips *all* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document.
- **Ignore ("ignorable")** - Strips all *ignorable* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
- **None ("none")** - Strips *no* whitespace before further processing.
- **Optimization level ("-opt")** - Set the optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization). This option allows optimization to be suppressed in cases where reducing the compiling time is important, where optimization conflicts with debugging, or causes extension functions with side-effects to behave unpredictably.
- **Use linked tree model ("-tree:linked")** - This option activates the linked tree model.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - If checked, Saxon runs the XQuery transformation with the XQuery 3.0 support (this option is enabled by default).

The following option is available for Saxon 9.6.0.5 Professional Edition (PE) and Enterprise Edition (EE) only:

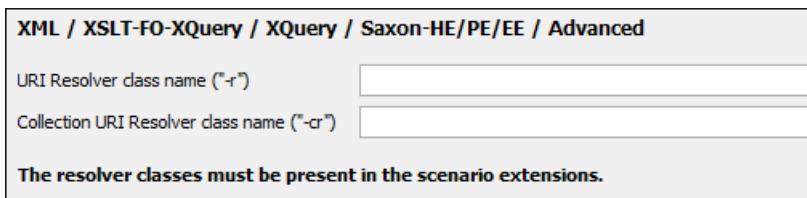
- **Allow calls on extension functions ("-ext")** - If checked, calls on external functions are allowed. Checking this option is recommended in an environment where untrusted stylesheets may be executed. It also disables user-defined extension elements and the writing of multiple output files, both of which carry similar security risks.

The options available specifically for Saxon 9.6.0.5 Enterprise Edition (EE) are as follows:

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and enables parsing the source documents with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.
- **Generate bytecode ("--generateByteCode:(on|off)")** - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to <http://www.saxonica.com/documentation9.5/index.html#!javadoc>.
- **Enable XQuery update ("-update:(on|off)")** - This option controls whether or not XQuery update syntax is accepted.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, backup versions for any XML files updated with XQuery Update are generated. This option is available when the **Enable XQuery update** option is enabled.

#### Saxon HE/PE/EE Advanced Preferences

To configure **Saxon HE/PE/EE Advanced** preferences, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE > Advanced**.



**Figure 417: The Saxon HE/PE/EE XQuery Advanced Preferences Panel**

The advanced XQuery options which can be configured for the Saxon 9.6.0.5 XQuery transformer (all editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **URI Resolver class name** - Allows you to specify a custom implementation for the URI resolver used by the XQuery Saxon 9.6.0.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XQuery extension](#) for the particular transformation scenario.



**Note:** If your `URIResolver` implementation does not recognize the given resource, the `resolve(String href, String base)` method should return a `null` value. Otherwise, the given resource will not be resolved through the [XML catalog](#).

- **Collection URI Resolver class name** - Allows you to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.6.0.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XQuery extension](#) for the particular transformation scenario.

## Debugger Preferences

To configure the **Debugger** preferences, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > Debugger**.

The following preferences are available:

- **Show xsl:result-document output** - if checked, the debugger presents the output of `xsl:result-document` instructions into the debugger output view;
- **Infinite loop detection** - set this option to receive notifications when an infinite loop occurs during transformation;
- **Maximum depth in templates stack** - sets how many `xsl:template` instructions can appear on the current stack. This setting is used by the infinite loop detection;
- **Debugger layout** - a horizontal layout means that the stack of XML editors takes the left half of the editing area and the stack of XSL editors takes the right one. A vertical layout means that the stack of XML editors takes the upper half of the editing area and the stack of XSL editors takes the lower one;
- **Debugger current instruction pointer** - controls the background color of the current execution node, both in the document (XML) and XSLT/XQuery views;
- **XWatch evaluation timeout (seconds)** - specifies the maximum time that Oxygen XML Editor allocates to the evaluation of XPath expressions while debugging;
- **Messages** - specifies whether a debugging session is stopped, is continued, or you are asked what to do when you are editing the source document involved in a debugging session.

## Profiler Preferences

This section explains the settings available for the XSLT Profiler. To access and modify these settings, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > Profiler** (see [Debugger Preferences](#) on page 1057).

The following profiles settings are available:

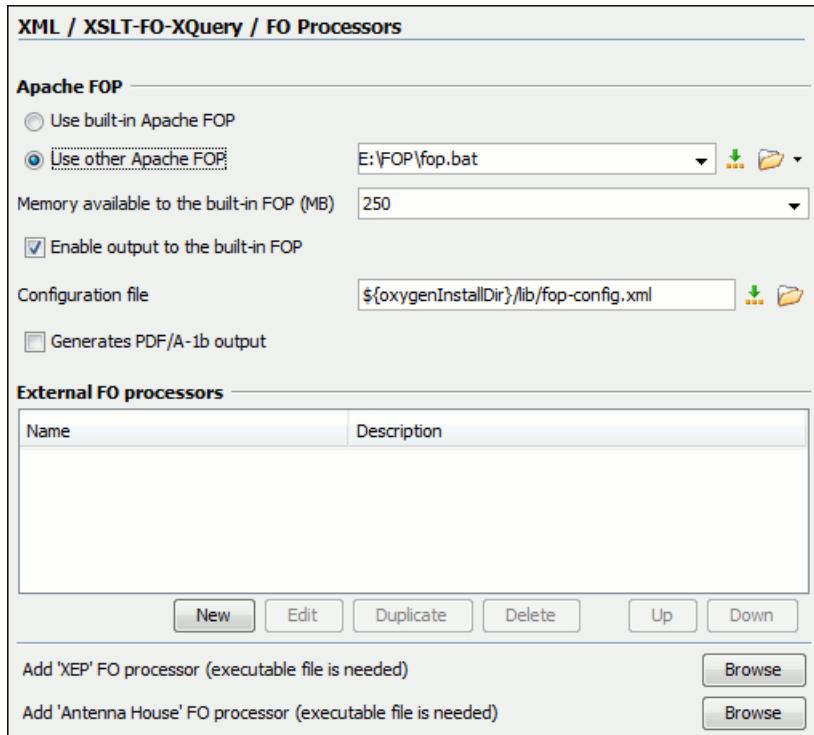
- **Show time** - Shows the total time that was spent in the call.
- **Show inherent time** - Shows the inherent time that was spent in the call. The inherent time is defined as the total time of a call minus the time of its child calls.
- **Show invocation count** - Shows how many times the call was called in this particular call sequence.
- **Time scale** - The time scale options determine the unit of time measurement, which may be milliseconds or microseconds.
- **Hotspot** threshold - The threshold below which hot spots are ignored (milliseconds).
- **Ignore invocation less than** - The threshold below which invocations are ignored (microseconds).
- **Percentage calculation** - The percentage base determines against what time span percentages are calculated:
  - **Absolute** - Percentage values show the contribution to the total time.
  - **Relative** - Percentage values show the contribution to the calling call.

## FO Processors Preferences

Besides Apache FOP, the built-in formatting objects processor, you can configure other external processors and set them in the transformation scenarios for processing XSL-FO documents.

Oxygen XML Editor provides an easy way to add two of the most used commercial FO processors: *RenderX XEP* and *Antenna House XSL Formatter*. You can easily add RenderX XEP as an external FO processor if you have the XEP installed. Also, if you have the Antenna House XSL Formatter v4 or v5, Oxygen XML Editor uses the environmental variables set by the XSL Formatter installation to detect and use it for XSL-FO transformations. If the environmental variables are not set for the XSL Formatter installation, you can browse and choose the executable file just as you would for XEP. You can use these two external FO processors for [DITA OT transformations scenarios](#) and [XML with XSLT transformation scenarios](#).

To configure the **FO Processors** options, [open the Preferences dialog box](#) and go to **XML > XSLT/FO/XQuery > FO Processors**.



**Figure 418: The FO Processors Preferences Panel**

### Apache FOP

The options for FO processors are the following:

- **Use built-in Apache FOP** - instructs Oxygen XML Editor to use its built-in Apache FO processor.
- **Use other Apache FOP** - instructs Oxygen XML Editor to use another Apache FO processor installed on your computer.
- **Enable the output of the built-in FOP** - all Apache FOP output is displayed in a results pane at the bottom of the Oxygen XML Editor window including warning messages about FO instructions not supported by Apache FOP.
- **Memory available to the built-in FOP** - if your Apache FOP transformations fail with an Out of Memory error (**OutOfMemoryError**) select from this combo box a larger value for the amount of memory reserved for FOP transformations.
- **Configuration file for the built-in FOP** - you should specify here the path to an Apache FOP configuration file, necessary for example to render to PDF a document containing Unicode content using a special *true type* font.
- **Generates PDF/A-1b output** - when selected PDF/A-1b output is generated.

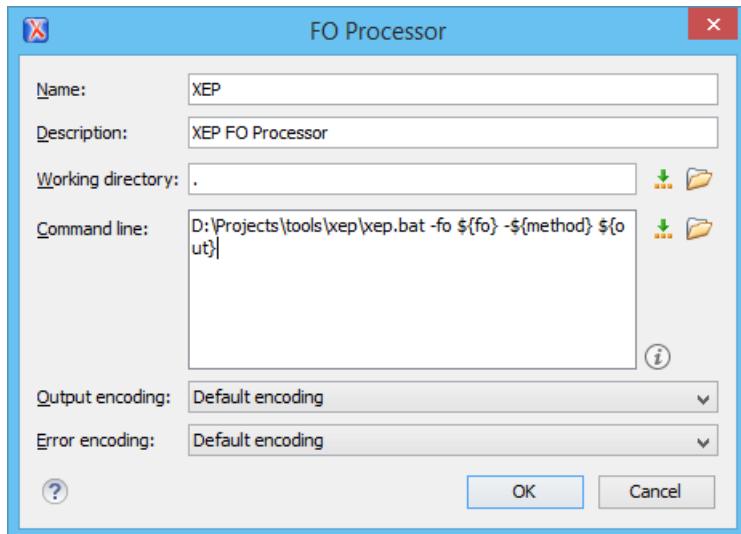
**Note:** All fonts have to be embedded, even the implicit ones. More information about configuring metrics files for the embedded fonts can be found in [Add a font to the built-in FOP](#).

**Note:** You cannot use the `<filterList>` key in the configuration file because FOP would generate the following error: *The Filter key is prohibited when PDF/A-1 is active*.

- **Add 'XEP' FO processor (executable file is needed)** - in case *RenderX XEP* is already installed on your computer, you can use this button to choose the XEP executable script (*xep.bat* for Windows, *xep* for Linux).
- **Add 'Antenna House' FO processor (executable file is needed)** - in case *Antenna House XSL Formatter* is already installed on your computer, you can use this button to choose the Antenna House executable script (*AHFCmd.exe*,  
, or *XSLCmd.exe* for Windows, *AHFCmd.sh*, or *XSLCmd.sh* for Linux).

### External FO processors

In this section you can manage the external FO processors you want to use in transformation scenarios. Press the **New** button to add a new external FO processor. The following dialog is displayed:



**Figure 419: The External FO Processor Configuration Dialog**

- **Name** - the name displayed in the list of available FOP processors on the FOP tab of the transformation scenario dialog.
- **Description** - a textual description of the FO processor displayed in the FO processors table and in tooltips of UI components where the processor is selected.
- **Working directory** - the directory where the intermediate and final results of the processing is stored. Here you can use one of the following editor variables:
  - **\${homeDir}**  - the path to user home directory.
  - **\${cfid}**  - the path of current file directory. If the current file is not a local file, the target is the user's desktop directory.
  - **\${pd}**  - the project directory.
  - **\${oxygenInstallDir}**  - the Oxygen XML Editor installation directory.
- **Command line** - the command line that starts the FO processor, specific to each processor. Here you can use one of the following editor variables:
  - **\${method}**  - the FOP transformation method: **pdf**, **ps** or **txt**
  - **\${fo}**  - the input FO file
  - **\${out}**  - the output file
  - **\${pd}**  - the project directory
  - **\${frameworksDir}**  - the path of the **frameworks** subdirectory of the Oxygen XML Editor install directory
  - **\${oxygenInstallDir}**  - the Oxygen XML Editor installation directory
  - **\${ps}**  - the platform-specific path separator. It is used between the library files specified in the class path of the command line
- **Output Encoding** - the encoding of the FO processor output stream displayed in a results panel at the bottom of the Oxygen XML Editor window.

- **Error Encoding** - the encoding of the FO processor error stream displayed in a results panel at the bottom of the Oxygen XML Editor window.

## XPath Preferences

To configure the **XPath** options, [open the \*\*Preferences\*\* dialog box](#) and go to **XML > XSLT/FO/XQuery > XPath**.

Oxygen XML Editor allows you to customize the following options:

- **Unescape XPath expression** - the entities of an XPath expressions that you type in the XPath/XQuery Builder and the [XPath toolbar](#) are unescaped during their execution. For example the expression

```
//varlistentry[starts-with(@os, 's')]
```

is equivalent with:

```
//varlistentry[starts-with(@os, 's')]
```

- **Multiple XPath results** - enable this option to display the results of an XPath in separate tabs in the [The Results View](#) on page 87.
- **No namespace** - if you enable this option, Oxygen XML Editor considers unprefixed element names of the evaluated XPath 2.0 / 3.0 expressions as belonging to no namespace.
- **Use the default namespace from the root element** - if you enable this option, Oxygen XML Editor considers unprefixed element names of the evaluated XPath expressions as belonging to the default namespace declared on the root element of the XML document you are querying.
- **Use the namespace of the root** - if you enable this option, Oxygen XML Editor considers unprefixed element names of the evaluated XPath expressions as belonging to the same namespace as the root element of the XML document you are querying.
- **This namespace** - in this field you can enter the namespace of the unprefixed elements.
- **Default prefix-namespace mappings** - in this field you can associate prefixes with namespaces. Use these mappings when you want to define them globally, not for each document.

## Custom Engines Preferences

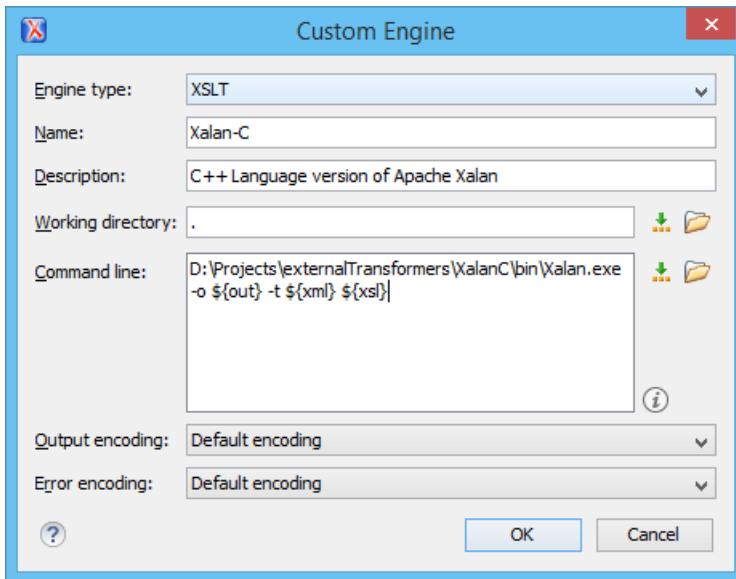
You can configure and run XSLT and XQuery transformations with processors other than [the ones which come with the Oxygen XML Editor distribution](#).



**Note:** You can not use these custom engines in [the Debugger perspective](#).

To configure the **Custom Engines** preferences, [open the \*\*Preferences\*\* dialog box](#) and go to **XML > XSLT/FO/XQuery > Custom Engines**.

The following parameters can be configured for a custom engine:



**Figure 420: Parameters of a Custom Engine**

- **Engine type** - Combo box allowing you to choose the transformer type. There are two options: XSLT engines and XQuery engines.
- **Name** - The name of the transformer displayed in the dialog for editing transformation scenarios
- **Description** - A textual description of the transformer.
- **Working directory** - The start directory of the transformer executable program. The following editor variables are available for making the path to the working directory independent of the location of the input files:
  - **`\${homeDir}`** - The user home directory in the operating system.
  - **`\${cfid}`** - The path to the directory of the current file.
  - **`\${pd}`** - The path to the directory of the current project.
  - **`\${oxygenInstallDir}`** - The Oxygen XML Editor install directory.
- **Command line** - The command line that must be executed by Oxygen XML Editor to perform a transformation with the engine. The following editor variables are available for making the parameters in the command line (the transformer executable, the input files) independent of the location of the input files:
  - **`\${xml}`** - The XML input document as a file path.
  - **`\${xmlu}`** - The XML input document as a URL.
  - **`\${xsl}`** - The XSL / XQuery input document as a file path.
  - **`\${xslu}`** - The XSL / XQuery input document as a URL.
  - **`\${out}`** - The output document as a file path.
  - **`\${outu}`** - The output document as a URL.
  - **`\${ps}`** - The platform separator which is used between library file names specified in the class path.
- **Output Encoding** - The encoding of the transformer output stream.
- **Error Encoding** - The encoding of the transformer error stream.

## Ant Preferences

To set Ant preferences, [open the Preferences dialog box](#) and go to **XML > Ant**. This panel allows you to choose the directory containing the Apache Ant libraries (the so-called *Ant Home*) that Oxygen XML Editor uses to handle Ant build files.

There are two options available:

- **Built-in** - the path to the Ant distribution that comes bundled with Oxygen XML Editor installation kit.
- **Custom** - the path to an Ant distribution of your choice.

## Import Preferences

To configure the **Import** options, [open the Preferences dialog box](#) and go to **XML > Import**. This page allows you to configure how empty values and null values are handled when they are encountered in imported database tables or Excel sheets. Also you can configure the format of date / time values recognized in the imported database tables or Excel sheets.

The following options are available:

- **Create empty elements for empty values** - If checked, an empty value from a database column or from a text file is imported as an empty element.
- **Create empty elements for null values** - If checked, null values from a database column are imported as empty elements.
- **Escape XML content** - Enabled by default, this option instructs Oxygen XML Editor to escape the imported content to an XML-safe form.
- **Add annotations for generated XML Schema** - If checked, the generated XML Schema contains an annotation for each of the imported table columns. The documentation inside the annotation tag contains the remarks of the database columns (if available) and also information about the conversion between the column type and the generated XML Schema type.

The section **Date / Time Format** specifies the format used for importing date and time values from Excel spreadsheets or database tables and in the generated XML schemas. The following format types are available:

- **Unformatted** - If checked, the date and time formats specific to the database are used for import. When importing data from Excel a string representation of date or time values are used. The type used in the generated XML Schema is `xs:string`.
- **XML Schema date format** - If checked, the XML Schema-specific format ISO8601 is used for imported date / time data (`yyyy-MM-dd'T'HH:mm:ss` for `datetime`, `yyyy-MM-dd` for `date` and `HH:mm:ss` for `time`). The types used in the generated XML Schema are `xs:datetime`, `xs:date` and `xs:time`.
- **Custom format** - If checked, the user can define a custom format for timestamp, date, and time values or choose one of the predefined formats. A preview of the values is presented when a format is used. The type used in the generated XML Schema is `xs:string`.

## Date / Time Patterns Preferences

**Table 17: Pattern letters**

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am / pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am / pm (0-11)	Number	0

Letter	Date or Time Component	Presentation	Examples
h	Hour in am / pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- *Text* - If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available.
- *Number* - The number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount.
- *Year* - If the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number.
- *Month* - If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.
- *General time zone* - Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used:
  - *GMTOffsetTimeZone* - GMT Sign Hours : Minutes
  - *Sign* - one of + or -
  - *Hours* - one or two digits
  - *Minutes* - two digits
  - *Digit* - one of 0 1 2 3 4 5 6 7 8 9

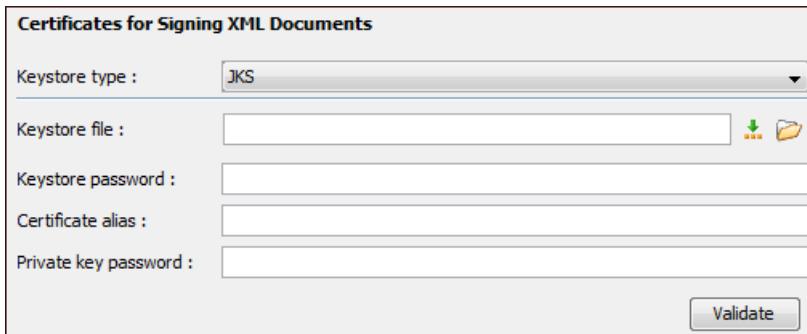
Hours must be between 0 and 23, and Minutes must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

- *RFC 822 time zone*: The RFC 822 4-digit time zone format is used:
  - *RFC822TimeZone* - Sign TwoDigitHours Minutes
  - *TwoDigitHours* - a number of two digits

TwoDigitHours must be between 00 and 23.

## XML Signing Certificates Preferences

Oxygen XML Editor provides two types of *keystores* for certificates that are used for digital signatures of XML documents: Java KeyStore (**JKS**) and Public-Key Cryptography Standards version 12 (**PKCS-12**). A *keystore* file is protected by a password. To configure a certificate *keystore*, [open the Preferences dialog box](#) and go to **XML > XML Signing Certificates**. You can customize the following parameters of a *keystore*:



**Figure 421: The Certificates Preferences Panel**

- **Keystore type** - The type of *keystore* that Oxygen XML Editor uses (**JKS** or **PKCS-12**).
- **Keystore file** - The location of the imported file.
- **Keystore password** - The password that is used for protecting the privacy of the stored keys.
- **Certificate alias** - The alias used for storing the key entry (the certificate or the private key) inside the *keystore*.
- **Private key password** - The private key password of the certificate (required only for JKS *keystores*).
- **Validate** - Press this button to verify the configured *keystore* and the validity of the certificate.

## XML Refactoring

To specify a folder for loading the custom XML refactoring operations, [open the Preferences dialog box](#) and go to **XML > XML Refactoring**. The following option is available in this preferences page:

- **Load additional refactoring operations from** - Use this text box to specify a folder for loading custom XML refactoring operations. You can use *editor variables* by pressing the **Insert Editor Variable** button or search for a folder by using the **Browse...** button.

## DITA Preferences

To access the DITA Preferences page, [open the Preferences dialog box](#) and click on **DITA**.

The **DITA-OT directory** option specifies the directory of the DITA Open Toolkit distribution to be used, by default, for validating and publishing DITA content. You can either provide a new file path for the specific DITA OT that you want to use or you can select a previously used one from the drop-down list.

**Note:** The DITA Open Toolkit is bundled with the Oxygen XML Editor installation.

The **DITA Maps file patterns** option allows you to set the extension of the files that will be handled as DITA maps when opened in Oxygen XML Editor. Oxygen XML Editor can open a DITA map in the regular editor view or in the [DITA Maps Manager](#).

The following options are available **When opening a map**:

- **Always open in the DITA Maps Manager** - A DITA Map file is always opened in the **DITA Maps Manager** view.
- **Always open as XML** - A DITA Map file is always opened in the XML editor.
- **Always ask** - When opening a DITA Map, you are prompted to choose between opening it in the XML editor panel or in the **DITA Maps Manager** view.

The **Show console output** option allows you to specify when to display the console output log. The following options are available:

- **When build fails** - displays the console output log if the build fails.
- **Always** - displays the console output log, regardless of whether or not the build fails.

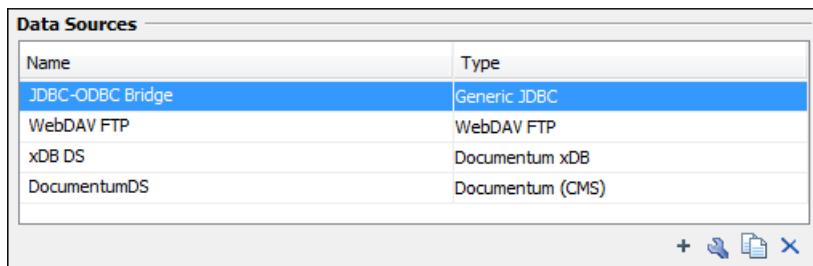
At the bottom of the page there is a link to the [Profiling Attributes preferences](#), where you can configure how profiling and conditional text is shown in **Author** mode.

## Data Sources Preferences

To configure the **Data Sources** preferences, [open the Preferences dialog box](#) and go to **Data Sources**.

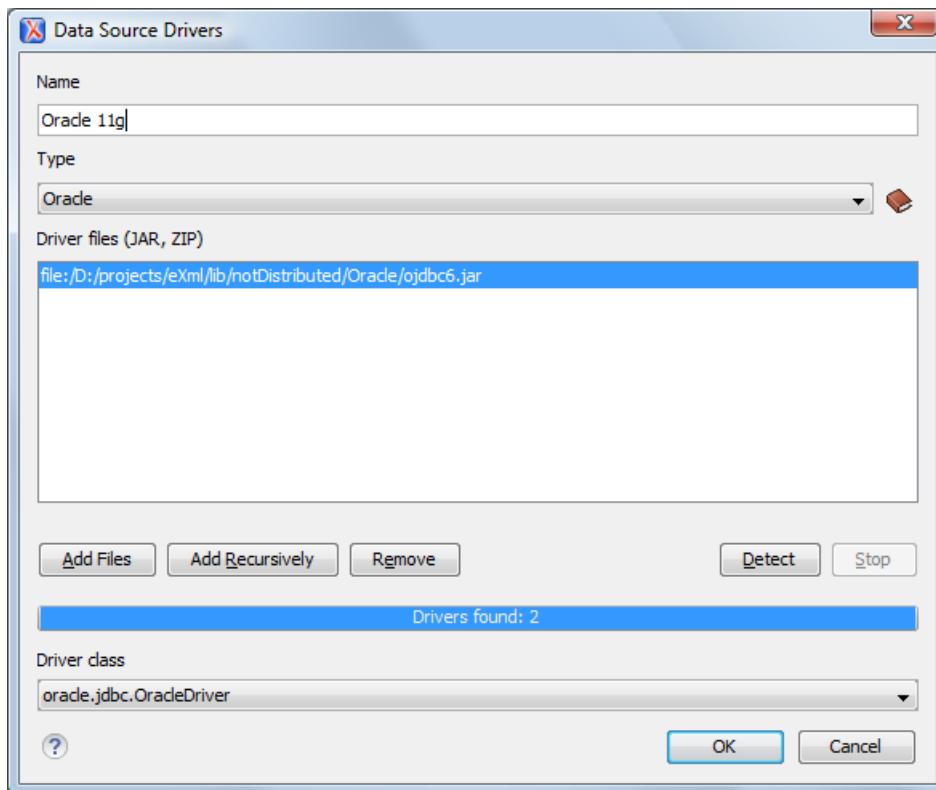
### Data Sources Preferences

To configure the **Data Sources** preferences, [open the Preferences dialog box](#) and go to **Data Sources**. In this preferences page you can configure data sources and connections to relational databases as well as native XML databases. You can check the list of drivers ([http://www.oxygenxml.com/database\\_drivers.html](http://www.oxygenxml.com/database_drivers.html)) available for the major database servers.



**Figure 422: The Data Sources Preferences Panel**

- **New** - opens the **Data Sources Drivers** dialog that allows you to configure a new database driver.

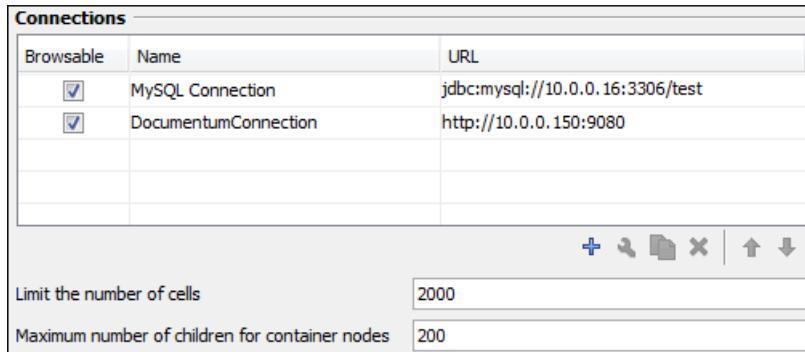


**Figure 423: The Data Sources Drivers Dialog**

The following options are available:

- **Name** - The name of the new data source driver that will be used for creating connections to the database.
- **Type** - Selects the data source type from the supported driver types.
- **Help** - Opens the User Manual at [the list of the sections](#) where the configuration of supported data sources is explained and the URLs for downloading the database drivers are specified.
- **Driver Class** - Specifies the driver class for the data source driver.
- **Add** - Adds the driver class library.

- **Remove** - Removes the selected driver class library from the list.
- **Detect** - Detects driver class candidates.
- **Stop** - Stops the detection of the driver candidates.
- **Edit** - Opens the **Data Sources Drivers** dialog for editing the selected driver. See above the specifications for the **Data Sources Drivers** dialog. In order to edit a data source, there must be no connections using that data source driver.
- **Delete** - Deletes the selected driver. In order to delete a data source, there must be no connections using that data source driver.



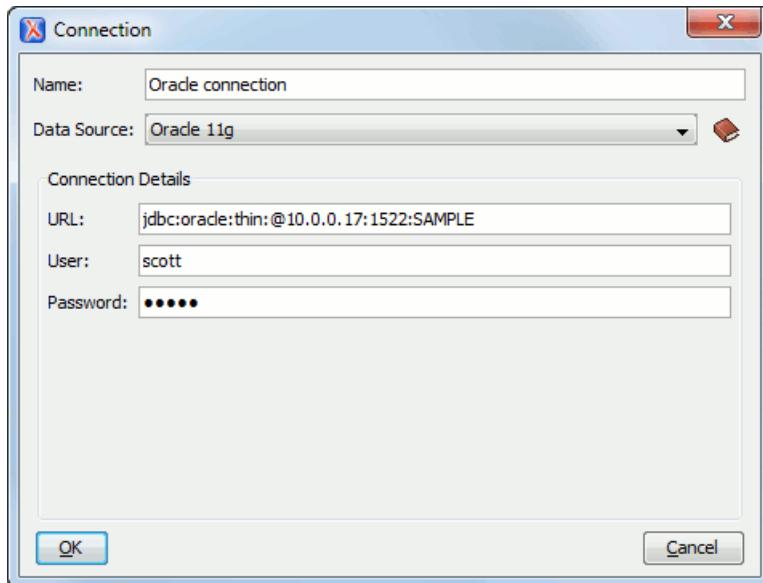
**Figure 424: The Connections Preferences Panel**

For performance issues, you can set the maximum number of cells that will be displayed in the **Table Explorer** view for a database table. Leave the field **Limit the number of cells** empty if you want the entire content of the table to be displayed. By default this field is set to 2,000. If a table having more cells than the value set here is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

In Oracle XML a container can hold millions of resources. If the node corresponding to such a container in the **Data Source Explorer** view would display all the contained resources at the same time the performance of the view would be very slow. To prevent such a situation only a limited number of the contained resources is displayed as child nodes of the container node. Navigation to other contained resources from the same container is enabled by the *Up* and *Down* buttons in the **Data Source Explorer** view. This limited number is set in the option **Maximum number of children for container nodes**. The default value is 200 nodes.

The actions of the buttons from the **Connections** panel are the following:

- **New** - opens the **Connection** dialog which has the following fields:



**Figure 425: The Connection Dialog**

- **Name** - The name of the new connection that will be used in transformation scenarios and validation scenarios.
- **Data Source** - Allows selecting a data source defined in the **Data Source Drivers** dialog.

Depending upon the selected data source, you can set some of the following parameters in the **Connection details** area:

- **URL** - The URL for connecting to the database server.
- **User** - The user name for connecting to the database server.
- **Password** - The password of the specified user name.
- **Host** - The host address of the server.
- **Port** - The port where the server accepts the connection.
- **XML DB URI** - The database URI.
- **Database** - The initial database name.
- **Collection** - One of the available collections for the specified data source.
- **Environment home directory** - Specifies the home directory (only for a Berkeley database).
- **Verbosity** - Sets the verbosity level for output messages (only for a Berkeley database).
- **Use a secure HTTPS connection (SSL)** - Allows you to establish a secure connection to an eXist database through the SSL protocol.
- **Edit** - Opens the **Connection** dialog, allowing you to edit the selected connection. See above the specifications for the **Connection** dialog.
- **Duplicate** - Creates a duplicate of the currently selected connection.
- **Delete** - Deletes the selected connection.

### Download Links for Database Drivers

Below you can find instructions for getting the drivers that are necessary to access databases in Oxygen XML Editor.

- **Berkeley DB XML database** - Copy the *.jar* files from the Berkeley database install directory into the Oxygen XML Editor install directory as described in [the procedure for configuring a Berkeley DB data source](#).
- **IBM DB2 Pure XML database** - Go to the [IBM website](#) and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill out the download form and download the zip file. Unzip the zip file and use the *db2jcc.jar* and *db2jcc\_license\_cu.jar* files in Oxygen XML Editor for [configuring a DB2 data source](#).
- **eXist database** - Copy the *.jar* files from the eXist database install directory to the Oxygen XML Editor install directory as described in [the procedure for configuring an eXist data source](#).

- **MarkLogic database** - Download the MarkLogic driver from [MarkLogic Community site](#).
- **Microsoft SQL Server 2005 / 2008 database** - Download the appropriate MS SQL JDBC driver from the Microsoft website. For SQL Server 2008 R2 and older go to <http://www.microsoft.com/en-us/download/details.aspx?id=21599>. For SQL Server 2012 and 2014 go to <http://www.microsoft.com/en-us/download/details.aspx?id=11774>.
- **Oracle 11g database** - Go to the [Oracle website](#) and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
- **PostgreSQL 8.3 database** - Go to the [PostgreSQL website](#) and download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar`.
- **Documentum xDb (X-Hive/DB) 10 XML database** - Copy the *jar* files from the Documentum xDb (X-Hive/DB) 10 database install directory to the Oxygen XML Editor install directory as described in [the procedure](#) for configuring a Documentum xDb (X-Hive/DB) 10 data source.

### Table Filters Preferences

to configure the **Table Filters** preferences, [open the Preferences dialog box](#) and go to **Data Sources > Table Filters**.

Here you can choose which of the database table types will be displayed in the **Data Source Explorer** view.

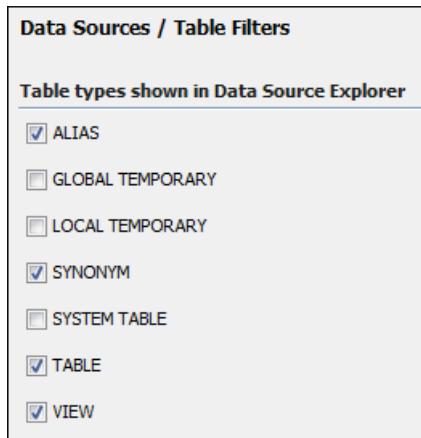
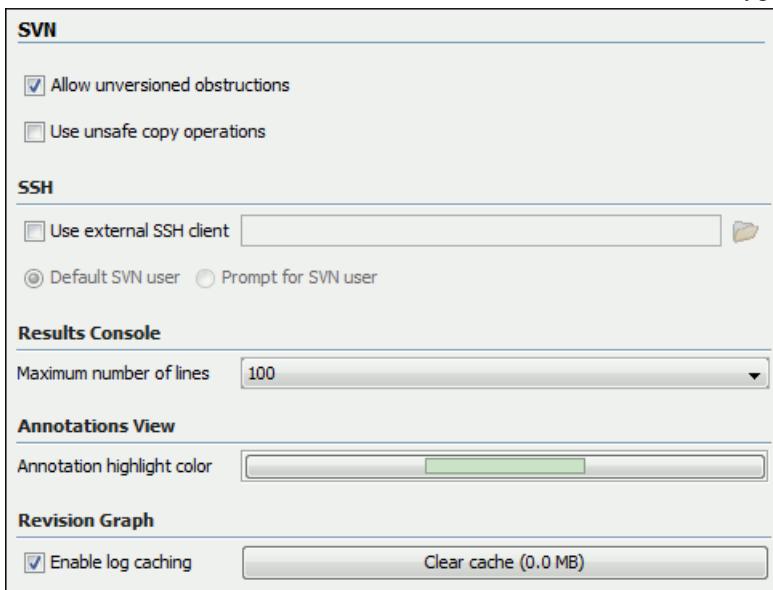


Figure 426: Table Filters Preferences Panel

### SVN Preferences

To configure the **SVN** options, [open the Preferences dialog box](#) and go to **SVN**. In this preferences page the user preferences for the embedded SVN client tool are configured. Some other preferences for the embedded SVN client tool can be set in the global files called `config` and `servers`, that is the files with parameters that act as defaults applied to all the SVN client tools that are used by the same user on his login account on the computer. These files can be opened for editing with the two edit actions available in the SVN client tool on the **Global Runtime Configuration** submenu of the **Options** menu.



**Figure 427: The SVN Preferences Panel**

The SVN preferences are the following:

- **Enable symbolic link support** (*available only on Mac OS X and Linux*) - Apache Subversion™ has the ability to put a symbolic link under version control, via the usual SVN add command. The Subversion repository has no internal concept of a symbolic link, it stores a versioned symbolic link as an ordinary file with a `svn:special` property attached. The SVN client (on Unix) sees the property and translates the file into a symbolic link in the working copy.



**Note:** Windows file systems have no symbolic links, so a Windows client won't do any such translation: the object appears as a normal file.

If the symbolic link support is disabled then the versioned symbolic links, on Linux and OS X, are supported in the same way as on Windows, that is a text file instead of symbolic link is created.



**Important:** It is recommended to disable symbolic links support if you do not have versioned symbolic links in your repository, because the SVN operations will work faster. However, you should not disable this option when you do have versioned symbolic links in repository. In that case a workaround would be to reference the working copy by its real path, not a path that includes a symbolic link.

- **Allow unversioned obstructions** - Controls how should be handled working copy resources being ignored / unversioned when performing an update operation and from the repository are incoming files with the same name, in the same location, that intersect with those being ignored / unversioned. If the option is enabled, then the incoming items will become BASE revisions of the ones already present in the working copy, and those present will be made versioned resources and will be marked as modified. Exactly as if the user first made the update operation and after that he / she modified the files. If the option is disabled, the update operation will fail when encountering files in this situation, possibly leaving other files not updated. By default, this option is enabled.
- **Use unsafe copy operations** - Sometimes when the working copy is accessed through Samba and SVN client cannot make a safe copy of the committed file due to a delay in getting write permission the result is that the committed file will be saved with zero length (the content is removed) and an error will be reported. In this case this option should be selected so that SVN client does not try to make the safe copy.
- **HTTPS encryption protocols** - Sets a specific encryption protocol to be used when the *application accesses a repository through HTTPS protocol*. You can choose one of the following values:
  - **SSLv3, TLSv1** (default value)
  - **SSLv3 only**
  - **TLSv1 only**

- **SSH** - Specifies the command line for an external SSH client which will be used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments (if any). Depending on the SSH client used and your SSH server configuration you may need to specify in the command line the user name and / or private key / passphrase. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example on Windows the following command line uses the `plink.exe` tool as external SSH client for connecting to the SVN repository with SVN+SSH:

```
C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
host_name_or_IP_address_of_SVN_server
```

- **Results Console** - Specifies the maximum number of lines displayed in the **Console** view. The default value is 1,000.
- **Annotations View** - Sets the color used for highlighting in the editor panel all the changes contributed to a resource by the revision selected in *the Annotations view*.
- **Revision Graph** - Enables caching for the action of computing a revision graph. When a new revision graph is requested one of the caches from the previous actions may be used which will avoid running the whole query again on the SVN server. If a cache is used it will finish the action much faster.

## Working Copy Preferences

To configure the **Working Copy** preferences, [open the Preferences dialog box](#) and go to **SVN > Working Copy**. The option that you are able to configure in this preferences page are specific to SVN working copies:

- **Working copies location** - Allows you to define a location where you keep your working copies. This location is automatically suggested when you checkout a new working copy.
- **Working copy administrative directory** - Allows you to customize the directory name where the svn entries are kept for each directory in the working copy.
- **When loading an old format working copy** - You can instruct Syncro SVN Client to do one of the following:
  - **Always ask** - You are notified when such a working copy is used and you are allowed to choose what action to be taken - to upgrade or not the format of the current working copy.
  - **Never upgrade** - Older format working copies are left untouched. No attempt to upgrade the format is made.

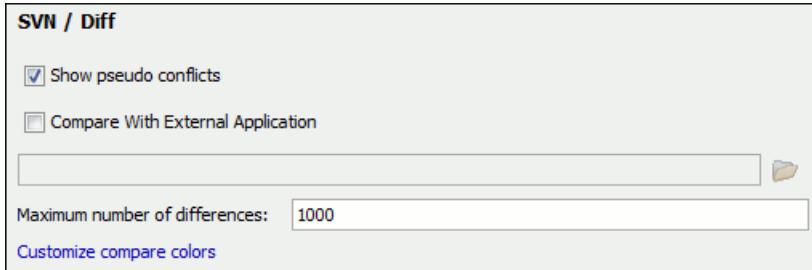


**Note:** SVN 1.6 and older working copies still need to be upgraded before loading them.

- **Enable working copy caching** - If checked, the content of the working copies is cached for refresh operations.
- **Automatically refresh the working copy** - If checked, the working copy is refreshed from cache. Only the new changes (modifications with a date/time that follows the last refresh operation) are refreshed from disk. Disabled by default.
- **Allow moving/rename mixed revision directories** - If enabled, the Syncro SVN Client will allow you to move or rename a directory even if its child items have a different revision. Otherwise, an error message is displayed when there are multiple revisions to avoid unnecessary conflicts. It is recommended to leave this option disabled and to **Update** the subtree to a single revision before moving or renaming it.
- **When synchronizing with repository** - The action that will be executed automatically after the **Synchronize** action. The possible actions are:
  - **Always switch to 'Modified' mode** - The **Synchronize** action is followed automatically by a switch to **Modified** mode of **Working Copy** view, if **All Files** mode is currently selected.
  - **Never switch to 'Modified' mode** - Keeps the currently selected view mode unchanged.
  - **Always ask** - The user is always asked if he wants to switch to **Modified** mode.
- **Application global ignores** - Allows setting file patterns that may include the \* and ? wildcards for unversioned files and folders that must be ignored when displaying the working copy resources in *the Working Copy view*. These patterns are case-sensitive. For example, \*.txt matches file.txt, but does not match file.TXT.

## Diff Preferences

To configure the SVN Diff options, [open the Preferences dialog box](#) and go to **Diff**.



**Figure 428: The SVN Diff Preferences Panel**

The SVN diff preferences are the following:

- **Show pseudo conflicts** - Specifies whether the [the Compare view](#) displays pseudo-conflicts . A pseudo-conflict occurs when two developers make the same change, for example when they both add or remove the same line of code.
- **Compare With External Application** - Specifies an external application to be launched for compare operations in the following cases:
  - When two history revisions are compared.
  - When the working copy file is compared with a history revision.
  - When [a conflict is edited](#).

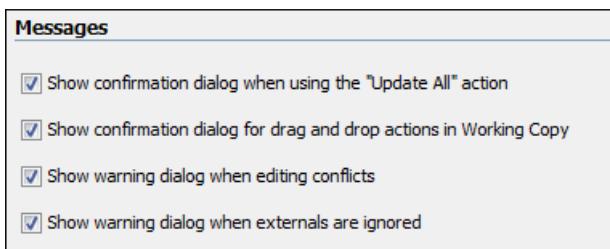
The parameters \${firstFile} and \${secondFile} specify the positions of the two compared files in the command line for the external diff application. The parameter \${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison. The working copy version of a file is compared with the repository version, with the BASE revision (the latest revision read from the repository by an Update or Synchronize operation) being the common ancestor of these two compared versions.

**Important:** If the path to the external compare application includes spaces (or any of the subsequent options or arguments), then each of these paths or *tokens* must be double-quoted for the Oxygen XML Editor to correctly parse and identify them. For example, C:\Program Files\compareDir\app name.exe must be written as "C:\Program Files\compareDir\app name.exe".

- **Maximum number of differences** - Sets the maximum number of differences allowed in the view.

## Messages Preferences

To configure the options for **Messages**, [open the Preferences dialog box](#) and go to **SVN > Messages**. This preferences page allows you to disable the following warning messages which may appear in the application:



**Figure 429: The Messages Preferences Panel**

- **Show confirmation dialog when using the "Update All" action** - Allows you to avoid performing accidental update operations by requesting you to confirm them before execution.
- **Show confirmation dialog for drag and drop actions in Working Copy** - This option avoids doing a drag and drop when you just want to select multiple files in the Working Copy view.
- **Show warning dialog when editing conflicts** - When the **Edit Conflicts** action is executed, a warning dialog notifies you that the action overwrites the conflicted version of the file created by an update operation. The conflicted file is

overwritten with the version of the same file which existed in the working copy before the update operation and then *proceeds with the visual editing of the conflicting file*.

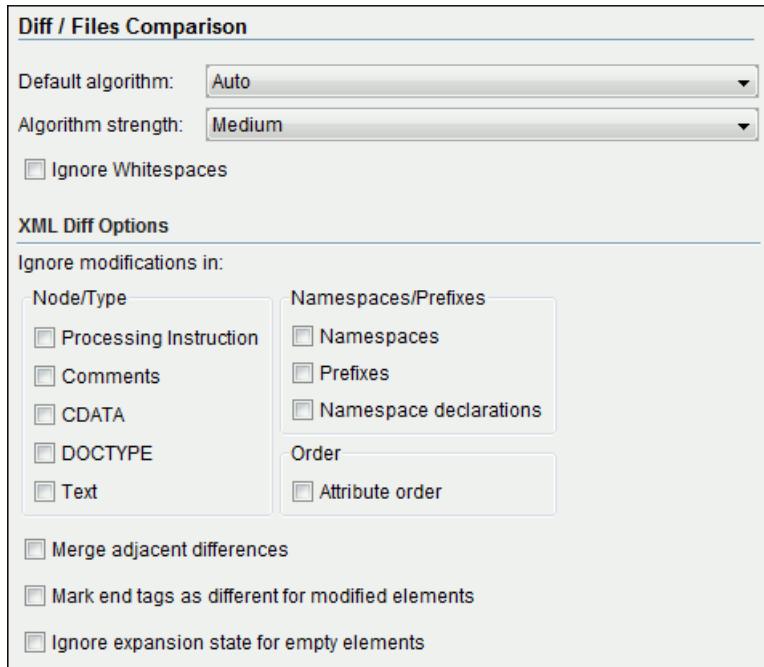
- **Show warning dialog when "svn:externals" definitions are ignored** - A warning dialog is displayed when "svn:externals" definitions are ignored before performing any operation that updates resources of the working copy (like *Update* and *Override and Update*).

## Diff Preferences

The Diff Preferences Page has sub-pages for configuring File Comparisons and Directory Comparisons.

### Files Comparison Preferences

To configure the **Files Comparison** options, [open the Preferences dialog box](#) and go to **Diff > Files Comparison**. This preferences page allows you to configure the following options:



**Figure 430: The Files Comparison Preferences Panel**

- **Default algorithm** - the default algorithm used for comparing files. The following options are available:
  - **Auto** - automatic selection of the diff algorithm, based on the content and size
  - **Characters** - computes the differences at character level
  - **Words** - computes the differences at word level
  - **Lines** - computes the differences at line level
  - **Syntax aware** - computes differences for the file types or XML fragments known by Oxygen XML Editor, taking the syntax (the specific types of tokens) into consideration
  - **XML Fast** - a diff algorithm well-suited for large XML documents or fragments (sacrifices accuracy in favor of speed)
  - **XML Accurate** - XML-tuned diff algorithm that favors accuracy over speed
- **Algorithm strength** - controls the amount of resources allocated to the application to perform the comparison. The algorithm stops searching more differences when reaches the maximum allowed resources. A dialog is shown when this limit is reached and partial results are displayed. Four settings are available: **Low**, **Medium** (default), **High** and **Very High**.
- **Ignore Whitespace** - ignoring whitespaces means that before performing the comparison, the application normalizes the content (collapses any sequence of whitespace characters (space, newline or tab characters) into a single space) and trims its leading and trailing whitespaces.

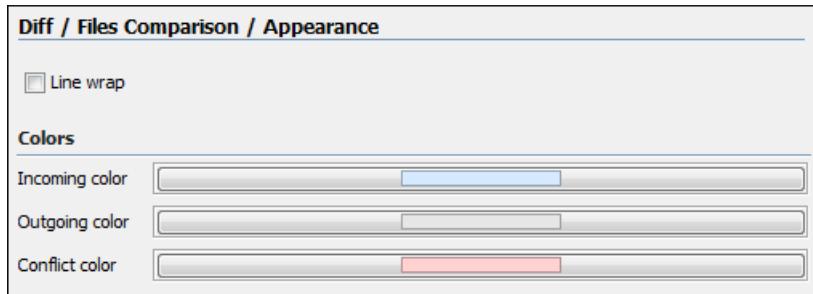


**Note:** If the **Ignore Whitespace** check box is selected, comparing the `a b` sequence with `a b`, Oxygen XML Editor finds no differences, because after normalization, all whitespaces from the first sequence are collapsed into a single space character. However, when comparing `a b` with `ab` (no whitespace between `a` and `b`), Oxygen XML Editor signals a difference.

- **XML Diff Options** - this set of options allows you to specify the types of XML nodes for which the differences will be ignored (will not be reported) by the **XML Fast** and **XML Accurate** algorithms.
- **Merge adjacent differences** - considers two adjacent differences as one when the differences are painted in the side-by-side editors. If unchecked, every difference is represented separately.
- **Mark end tags as different for modified elements** - end tags of modified elements are presented as differences too, otherwise only the start tags are presented as differences.
- **Ignore expansion state for empty elements** - empty elements in both expansion states are considered matched, that is `<element/>` and `<element></element>` are considered equal.

### Appearance Preferences

To configure the appearance options for the Files Comparison tool, [open the Preferences dialog box](#) and go to **Diff > Files Comparison > Appearance**. This preferences page offers the following options:



**Figure 431: Files Comparison Appearance Preferences Panel**

- **Line wrap** - Wraps at the right margin of each panel the lines presented in the two diff panels, so no horizontal scrollbar is necessary.
- **Incoming color** - Specifies the color used for incoming changes on the vertical bar, that shows the differences between the compared files.
- **Outgoing color** - Specifies the color used for outgoing changes on the vertical bar, that shows the differences between the compared files.
- **Conflict color** - Specifies the color used for conflicts on the vertical bar, that shows the differences between the compared files.

### Directories Comparison Preferences

To configure the **Directories Comparison** preferences, [open the Preferences dialog box](#) and go to **Diff > Directories Comparison**.



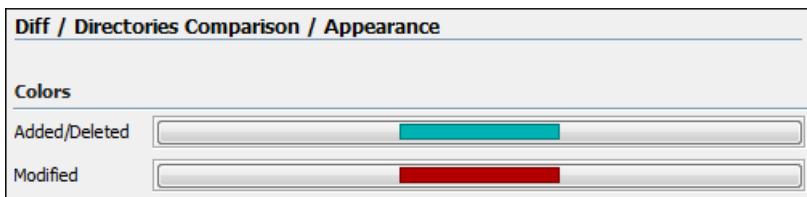
**Figure 432: The Diff Preferences Panel**

For the directories comparison, you can specify:

- **Compare files by** - Controls the method used for comparing two files:
  - **Content** - The file content is compared using the current *diff algorithm*. This option is applied for a pair of files only if that file type is associated with a built-in editor type (either associated by default or associated by the user when the user is prompted to do that on opening a file of that type for the first time).
  - **Binary Compare** - The files are compared at byte level.
  - **Timestamp (last modified date / time)** - The files are compared only by their last modified timestamp.
- **Look in archives** - If checked, *archives known by Oxygen XML Editor* are considered directories and their content is compared just like regular files.
- **Navigation** - This options control the behaviour of the differences traversal actions (**Go to previous modification**, **Go to next modification**) when the first or last difference in a file is reached:
  - **Ask what to do next** - A dialog is displayed asking you to confirm that you want the application to display modifications from the previous or next file.
  - **Go to the next/previous file** - The application opens the next or previous file without waiting for your confirmation.
  - **Do nothing** - No further action is taken.

### Appearance Preferences

To configure the appearance options for the Directories Comparison tool, [open the Preferences dialog box](#) and go to **Diff > Directories Comparison > Appearance**.



**Figure 433: The Diff Appearance Preferences Panel**

- **Added/Deleted** - Color used for marking added or deleted files and folders.
- **Modified** - Color used for marking modified files.

### Archive Preferences

To configure **Archive** preferences, [open the Preferences dialog box](#) and go to **Archive**.

The following options are available in the **Archive** preferences panel:

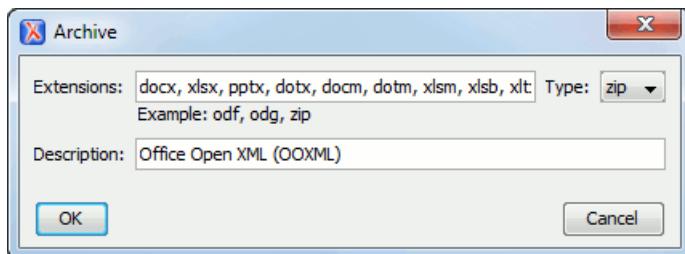
- **Archive backup options** - Controls if the application makes backup copies of the modified archives. The following options are available:

- **Always create backup copies of modified archives** - When you modify an archive, its content is backed up.
- **Never create backup copies of modified archives** - No backup copy is created.
- **Ask for each archive once per session** - Once per application session for each modified archive, user confirmation is required to create the backup. This is the default setting.



**Note:** Backup files have the name `originalArchiveFileName.bak` and are located in the same folder as the original archive.

- **Archive types** - This table contains all known archive extensions mapped to known archive formats. Each row maps a list of extensions to an archive type supported in Oxygen XML Editor. You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.



**Figure 434: Edit Archive Extension Mappings**



**Important:** You have to restart Oxygen XML Editor after removing an extension from the table in order for that extension to not be recognised anymore as an archive extension.

- **Store Unicode file names in Zip archives** - Use this option when you archive files that contain international (that is, non-English) characters in file names or file comments. If this option is selected and an archive is modified in any way, UTF-8 characters are used in the names of all files in the archive.

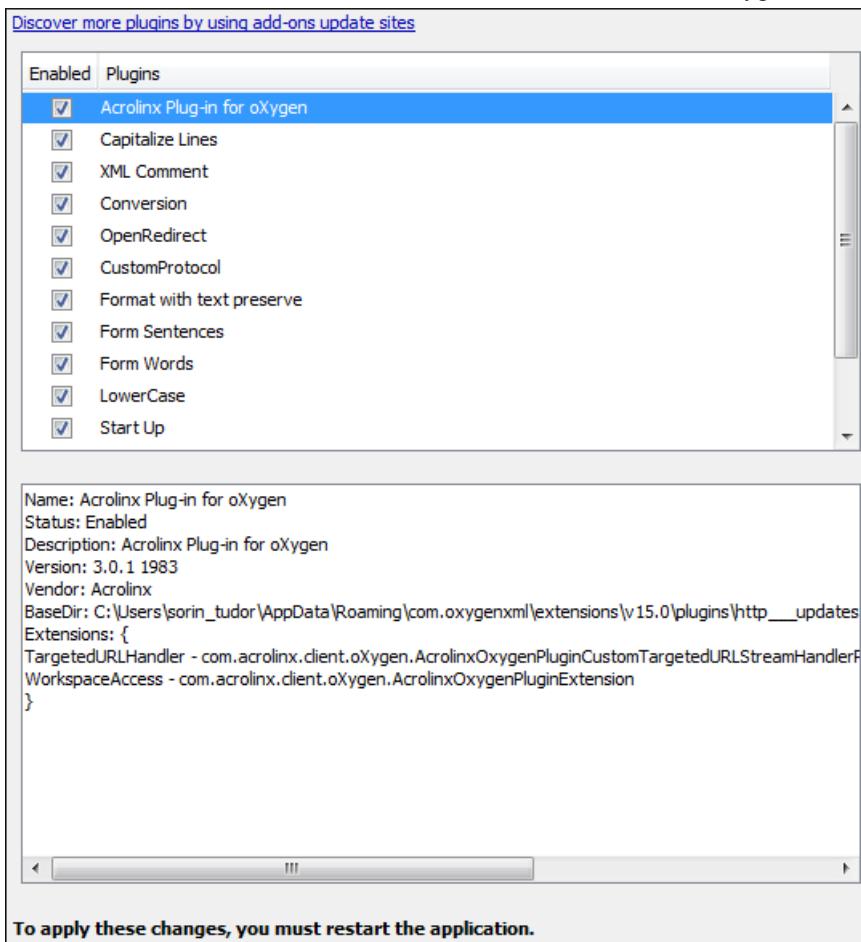
## Plugins Preferences

You are able to add plugins that extend the functionality of Oxygen XML Editor. The plugins are shipped as separate packages. To check for new plugins, go to [http://www.oxygenxml.com/oxygen\\_sdk.html](http://www.oxygenxml.com/oxygen_sdk.html).

A plugin consists of a separate sub-folder in the `Plugins` folder of the Oxygen XML Editor installation folder. This sub-folder must contain a valid `plugin.xml` file in accordance with the `plugin.dtd` file located in the `Plugins` folder.

Oxygen XML Editor automatically detects and loads plugins installed correctly in the `Plugins` folder and displays them in the **Plugins** preferences page. To configure plugins, [open the Preferences dialog box](#) and go to **Plugins**.

You can use the check-boxes in front of each plugin to enable or disable them. To display the properties of a plugin in the second section of the **Plugins** preferences page, click the name of the plugin.



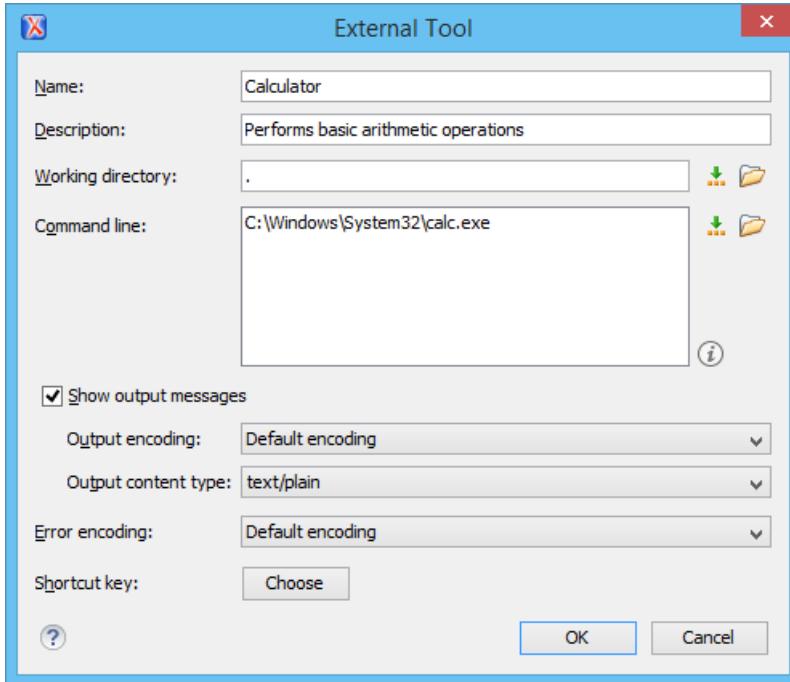
**Figure 435: The Plugins Preferences Panel**

Also, you are able to install a plugin as an add-on. For further details about this, go to [Deploying Add-ons](#)

## External Tools Preferences

The **External Tools** preferences panel is opened from menu **Tools > External Tools > Configure....**

A command-line tool can be started in the Oxygen XML Editor user interface as if from the command line of the operating system shell by selecting it from the **External Tools** drop-down list in the **Tools** toolbar. The tool must first be configured in the **External Tools** preferences page.



**Figure 436: The External Tools Configuration Dialog**

The configuration dialog includes the following options:

- **Name** - The name of the menu entry corresponding to this tool that will be displayed in the **Tools > External Tools** menu and in the **External Tools** drop-down list in the **Tools** toolbar.
- **Description** - The description of the tool displayed as tooltip where the tool name is used.
- **Working directory** - The directory the external tool will use to store intermediate and final results. Here you can use one of the following editor variables: `$(cfid)`, `$(pdid)`, `$(oxygenInstallDir)`, `$(homeDir)`, `$(system(var.name))`, `$(date(pattern))`, `$(xpath_eval(expression))`.
- **Command line** - The command line that will start the external tool. Here you can use one of the following editor variables: `$(homeDir)`, `$(home)`, `$(cfn)`, `$(cfne)`, `$(cf)`, `$(currentFileURL)`, `$(cfid)`, `$(cfdu)`, `$(tsf)`, `$(pd)`, `$(pdu)`, `$(oxygenInstallDir)`, `$(oxygenHome)`, `$(frameworksDir)`, `$(frameworks)`, `$(ps)`, `$(timeStamp)`, `$(uuid)`, `$(id)`, `$(afn)`, `$(afne)`, `$(af)`, `$(afu)`, `$(afd)`, `$(afdu)`, `$(ask('message', type, 'default_value'))`, `$(dbgXML)`, `$(dbgXSL)`, `$(env(VAR_NAME))`, `$(system(var.name))`, `$(date(pattern))`, and `$(xpath_eval(expression))`
- **Show output messages** - When this option is checked all the messages emitted by the external tool are displayed in the **Results** view. You can edit the following options:
  - **Output encoding** - The encoding of the output stream of the external tool that will be used by Oxygen XML Editor to read the output of the tool.
  - **Output content type** - A list of predefined content type formats that instruct Oxygen XML Editor how to display the generated output. For example, setting the **Output content type** to `text/xml` enables the syntax coloring of XML output.

When this option is unchecked only the error messages are displayed.

- **Error Encoding** - The encoding of the error stream of the external tool that will be used by Oxygen XML Editor to read this error stream.
- **Shortcut key** - The keyboard shortcut that launches the external tool.

## Menu Shortcut Keys Preferences

To configure the **Menu Shortcut Keys** options, [open the Preferences dialog box](#) and go to **Menu Shortcut Keys**.

Alternatively you can go to **Options > Menu Shortcut Keys**. You can use this page to configure shortcut keys for the actions available in Oxygen XML Editor. The shortcuts assigned to menu items are displayed in the below table.

Menu Shortcut Keys		
Description	Category	Shortcut key
Find Next	Find	F3
Find Previous	Find	Shift+F3
Find/Replace	Find	Ctrl+F
Find/Replace in Files	Find	Ctrl+Shift+H
Go to	Find	Ctrl+L
Quick Find	Find	Alt+Shift+F
Close Other Folds	Folding	Ctrl+NumPad /
Collapse Child Folds	Folding	Ctrl+NumPad .
Expand All	Folding	Ctrl+NumPad *
Expand Child Folds	Folding	
Toggle Fold	Folding	
Decrease Editor Font	Font Size	Ctrl+NumPad -
Increase Editor Font	Font Size	Ctrl+NumPad +
Normal Editor Font	Font Size	Ctrl+NumPad 0
Array	Grid	
Append value as child	Grid Append Child	
Attribute	Grid Append Child	
CDATA	Grid Append Child	
Comment	Grid Append Child	
Doctype	Grid Append Child	
Doctype identifier	Grid Append Child	
Doctype subset	Grid Append Child	
Element	Grid Append Child	

**Figure 437: The Menu Shortcut Keys Preferences Panel**

The **Menu shortcut Keys** preferences page also contains the shortcuts that you define at *document type* level.

 **Note:** A shortcut defined at *document type* level overwrites a default shortcut.

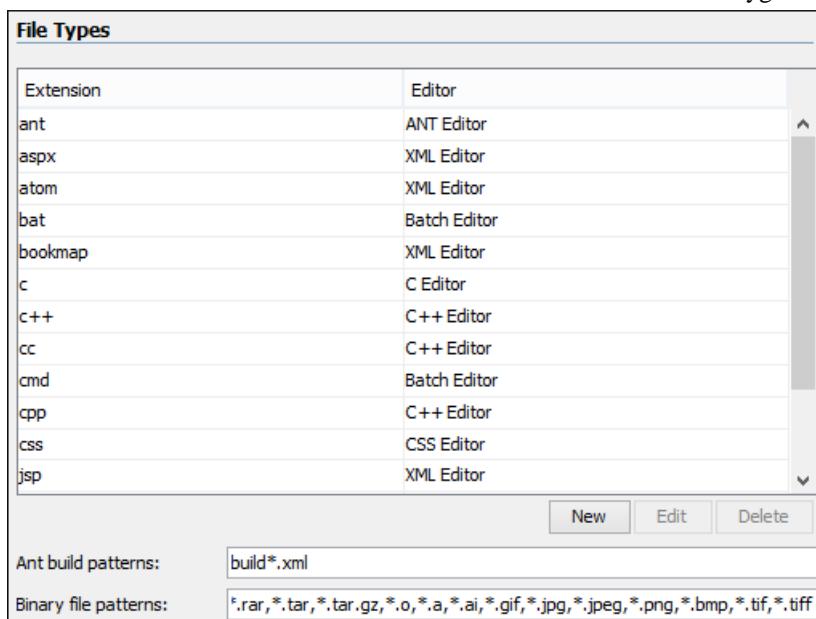
To find a specific action, you can use the filter to search through the **Description**, **Category**, and **Shortcut Key** columns:

- **Description** - this column provides a short description of the action.
- **Category** - this column provides a classification of the actions in categories for easier management. For example, you can distinguish the **Cut** operation available in the **Text** view from the one available in the **Tree** view by assigning them to different categories.
- **Shortcut key** - this column holds the combination of keys on your keyboard that launches the action. You can either double click a row of the **Shortcut key** column or press the **Edit** button to enter a new shortcut.
- **'Home' and 'End' keys are applied at line level** - Option available only on Mac OS X that controls the way the HOME and END keys are interpreted. If enabled, the default behavior of the Mac OS X HOME and END keys is overridden and the caret moves only on the current line.

## File Types Preferences

Oxygen XML Editor offers editing support for a wide variety of file types, but users are free to add new file extensions and associate them with the editor type which fits better. The associations set here between a file extension and the type of editor which opens the file for editing have precedence over the default associations available in [the File > New dialog](#).

To configure the **File Types** options, [open the Preferences dialog box](#) and go to **File Types**.



**Figure 438: The File Types Preferences Panel**

The table columns contain the following data:

- **Extension** - The extensions of the files that will be associated with an editor type.
- **Editor** - The type of editor which the extensions will be associated with. Some editors provide easy access to frequent operations via toolbars (e.g. XML editor, XSL editor, DTD editor) while others provide just a syntax highlight scheme (e.g. Java editor, SQL editor, Shell editor, etc.).

If the editor set here is not one of the XML editors (XML editor, XSL editor, XSD editor, RNG editor, WSDL editor) then the encoding set in [the preference Encoding for non XML files](#) is used for opening and saving a file of this type.

The files that match the **Ant build patterns** will be associated with the Ant editor.

The files that match the **Binary file patterns** patterns are handled as binary and opened in the associated system application. Also, they are excluded from the following actions available in the [Project view: File/Replace in Files, Check Spelling in Files, Validate](#).

## The Open/Find Resources Preferences Page

To configure the **Open/Find Resource** options, [open the Preferences dialog box](#) and go to **Open/Find Resource**.

The following options are available in the **Open/Find Resource** preferences page:

- **Limit search results to** - specifies the maximum number of results that are displayed in the **Open/Find Resource** dialog or in the **Open/Find Resources** view.
- **Enable searching in content** - enable this option to activate the **In content** option of the **Open/Find Resource** view and **Open/Find Resource** dialog. In case this option is disabled you can only use the **Open/Find Resource** feature to search in file paths.
- **Ignore content of these files** - allows you to select specific directories, files, or file types that are ignored when you perform a search. For example, `*.txt` ignores all the `.txt` files, `*/topics/*` ignores all the files from the `topics` directory, regardless of their depth, and `file:/C:/tmp/*` ignores everything from the `tmp` directory.



**Note:** The specified pattern must begin with the desired protocol (in our case `file`) and also contain forward slash (/).

- **Index the content of remote resources** - controls the indexing of resources that are not local. For example, the resources referenced in a DITA Map opened from a remote server (from a CMS or from a WebDAV location) are not indexed by default. To index the content of these resources, enable this option.



**Note:** Enabling this option may lead to delays when the indexing is computed.

- **Exact matches** - this option controls whether the search results should exactly match the whole words that you introduce in the search field of the **Open/Find Resource** dialog or **Open/Find Resources** view.
- **Prefix matches** - this option controls whether the search results should match documents containing words starting with the searched terms. For instance searching for "pref page" will find documents containing also "preference pages".
- **Automatically join search terms using:** - select the default boolean operator that Oxygen XML Editor applies when you perform a search. For example if the AND operator is selected and you enter "car assembly" in the dialog, the resulted documents must contain both of the words. If you choose OR, the resulted documents must contain one of the selected search terms. Results containing both words are promoted to the top of the list.
- **Enable XML-aware searching for files with size less than** - allows you to perform an *XML specific search* for XML elements and attributes.



**Note:** Enabling this option may slow down the indexing of your documents and increase the index size on the disk.

- **Stop Words** - list of comma separated stop words. Words added in this list are filtered out prior to processing a search query.

## Custom Editor Variables Preferences

An editor variable is useful for making a transformation scenario, a validation scenario or an external tool independent of the file path on which the scenario / command line is applied. An editor variable is specified as a parameter in a transformation scenario, validation scenario or command line of an external tool. Such a variable is defined by a name, a string value and a text description. A custom editor variable is defined by the user and can be used in the same expressions as the built-in ones.

Custom Editor Variables		
Name	Value	Description
<code> \${startDir}</code>	<code>../..../bin</code>	Start directory of command line validator
<code> \${standardParams}</code>	<code>-c config.xml -v -level 5 -list</code>	List of command line standard parameters

Figure 439: Custom Editor Variables

## Network Connection Settings Preferences

This section presents the options available in the **Network Connection Settings** preferences pages.

### Proxy Preferences

Some networks use proxy servers to provide internet services to LAN clients. Clients behind the proxy may therefore, only connect to the Internet via the proxy service. If you are not sure whether your computer is required to use a proxy server to connect to the Internet or you do not know the proxy parameters, consult your network administrator.

To configure the **Proxy** options, [open the Preferences dialog box](#) and go to **Network Connection Settings > Proxy**.

The following options are available:

- **Direct connection** - specifies whether the HTTP(S) connections go directly to the target host without going through a proxy server.
- **Use system settings** - specifies whether the HTTP(S) connections go through the proxy server set in the operating system. For example, on Windows the proxy settings are the ones that Internet Explorer uses.

 **Attention:** The system settings for the proxy cannot be read correctly from the operating system on some Linux systems. The system settings option should work properly on Gnome based Linux systems, but it does not work on KDE based ones as the Java virtual machine does not offer the necessary support yet.

- **Manual proxy configuration** - specifies whether the HTTP(S) connections go through the proxy server specified in the **Address** and **Port** fields.
- **No proxy for** - specifies the hosts to which the connections must not go through a proxy server. A host needs to be written as a fully qualified domain name (e.g. *myhost.example.com*) or as a domain name (e.g. *example.com*). Use comma as a separator between multiple hosts.
- **User** - specifies the user necessary for authentication with the proxy server.
- **Password** - specifies the password necessary for authentication with the proxy server.
- **SOCKS Proxy** - In this section you set the host and port of a SOCKS proxy through which all the connections pass. If the **Address** field is empty the connections use no SOCKS proxy.

### Using an automatic proxy configuration script (PAC)

If you have set up the path to an automatic proxy configuration script in your system (IE for Windows) Oxygen XML Editor cannot detect this setting out of the box.

You can create a new folder `[OXYGEN_DIR]\lib\endorsed` in which you should copy two additional Java libraries: `deploy.jar` and `plugin.jar`. These libraries can be found in the folder `[OXYGEN_DIR]\jre\lib` if the application comes with a bundled Java VM or otherwise in the Java VM installation used to run the application.

### HTTP(S)/WebDAV Preferences

To set the **HTTP(S)/WebDAV** preferences, [open the Preferences dialog box](#) and go to **Network Connection Settings > HTTP(S)/WebDAV**. The following options are available:

- **Internal Apache HttpClient Version** - Oxygen XML Editor uses the Apache HttpClient to establish connections to HTTP servers. To enable Oxygen XML Editor to benefit from particular sets of features provided by different versions, you may choose between v3 and v4.
 

 **Note:** For a full list of features, go to <http://hc.apache.org/httpclient-3.x/> and <http://hc.apache.org/httpcomponents-client-ga/>
- **Maximum number of simultaneous connections per host** - Defines the maximum number of simultaneous connections established by the application with a distinct host. Servers might consider multiple connections opened from the same source to be a **Denial of Service** attack. You can avoid that by lowering the value of this option.
 

 **Note:** This option accepts a minimum value of 5.
- **Read Timeout (seconds)** - The period in seconds after which the application considers that an HTTP server is unreachable if it does not receive any response to a request sent to that server.
- **Enable HTTP 'Expect: 100-continue' handshake for HTTP/1.1 protocol** - Activates *Expect: 100-Continue* handshake. The purpose of the *Expect: 100-Continue* handshake is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request headers) before the client sends the request body. The use of the *Expect: 100-continue* handshake can result in noticeable performance improvement when working with databases. The *Expect: 100-continue* handshake should be used with caution, as it may cause problems with HTTP servers and proxies that do not support the HTTP/1.1 protocol.

- **Use the 'Content-Type' header field to determine the content type** - When checked, Oxygen XML Editor tries to determine a resource type using the **Content-Type** header field. This header indicates the *Internet media type* of the message content, consisting of a type and subtype, for example:

```
Content-Type: text/xml
```

When unchecked, the resource type is determined by analyzing its extension. For example, a file ending in `.xml` is considered to be an XML file.

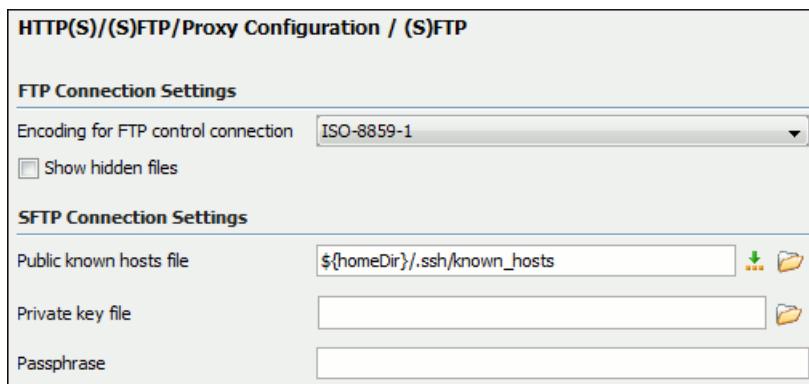
- **Automatic retry on recoverable error** - If enabled, if an HTTP error occurs when Oxygen XML Editor communicates with a server via HTTP, for example sending / receiving a SOAP request / response to / from a Web services server, and the error is recoverable, Oxygen XML Editor tries to send again the request to the server.
- **Automatically accept a security certificate, even if invalid** - When enabled, the HTTPS connections that Oxygen XML Editor attempts to establish will accept all security certificates, even if they are invalid.

 **Important:** By accepting an invalid certificate, you accept at your own risk a potential security threat, since you cannot verify the integrity of the certificate's issuer.

- **Encryption protocols (SVN Client only)** - this option is available only if you run the application with Java 1.6 or older. Sets a specific encryption protocol used when a repository is accessed through HTTPS protocol. You can choose one of the following values:
  - **SSLv3, TLSv1** (default value);
  - **SSLv3 only;**
  - **TLSv1 only.**
- **Lock WebDAV files on open** - If checked, the files opened through WebDAV are locked on the server so that they cannot be edited by other users while the lock placed by the current user still exists on the server.

## (S)FTP Preferences

To configure the (S)FTP options, [open the Preferences dialog box](#) and go to **Network Connection Settings > (S)FTP**. You can customize the following options:



**Figure 440: The (S)FTP Configuration Preferences Panel**

- **Encoding for FTP control connection** - The encoding used to communicate with FTP servers: either ISO-8859-1 or UTF-8. If the server supports the UTF-8 encoding Oxygen XML Editor will use it for communication. Otherwise it will use ISO-8859-1.
- **Public known hosts file** - File containing the list of all SSH server host keys that you have determined are accurate. The default value is  `${homeDir} / .ssh/known_hosts`.
- **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol. The user / password method of authentication has precedence if it is used in [the Open URL dialog](#).
- **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol. The user / password method of authentication has precedence if it is used in [the Open URL dialog](#).

## SSH Preferences

To configure the **SSH** options, [open the Preferences dialog box](#) and go to **Connection settings > SSH**. The following options are available:

- **Use external SSH client** - Allows you to specify the command line for an external SSH client which is used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments (if any). Depending on the SSH client used and your SSH server configuration, specify the user name and / or private key / passphrase in the command line. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example, on Windows the following command line uses the plink .exe tool as external SSH client for connecting to the SVN repository with SVN+SSH:

```
C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
host_name_or_IP_address_of_SVN_server
```

## XML Structure Outline Preferences

To configure the **XML Structure Outline** options, [open the Preferences dialog box](#) and go to **XML Structure Outline**, which contains the following preferences:

- **Preferred attribute names for display** - The preferred attribute names when displaying the attributes of an element in the **Outline** view. If there is no preferred attribute name specified, the first attribute of an element is displayed.
- **Enable outline drag and drop** - Drag and drop is disabled for the tree displayed in the **Outline** view only if there is a possibility to accidentally change the structure of the document by such operations.

## Views Preferences

To configure the view options, [open the Preferences dialog box](#) and go to **Views** and contains the following preferences:

- **Enable drag-and-drop in Project view** - Enables the drag and drop support in **Project** view. It should be disabled only if there is a possibility to accidentally change the structure of the project by such drag and drop actions.
- **Format and indent on save** - If selected, Oxygen XML Editor will run the action **Format and Indent** (pretty-print) on saving a document edited in the tree editor.
- **Maximum number of lines** - Sets the maximum number of lines in the **Information** view.

## Messages Preferences

To configure the **Messages** options, [open the Preferences dialog box](#) and go to **Messages**. This preferences page allows you to disable the following warning messages which may appear in the application:

- **Show confirmation dialog when moving resources** - displays a confirmation dialog box when you move a resource in the Project view, DB Explorer, and Archive Browser. In the **Confirm** dialog box you are able to choose not to see this dialog in the future.
- **Show warning when adding resources already included in the project** - displays a dialog box that warns you in case you try to add already existing files in your project.
- **Show warning for document size limit for bidirectional text, Asian languages, and other special characters** - displays a warning dialog box when the opened file that contains bidirectional characters is too large and bidirectional support is disabled.
- **Show warning message when changing the text orientation in the editor** - displays a warning dialog box when you change the text orientation in the editor.
- **Show warning when editing long expressions in the XPath toolbar** - displays an information dialog box that asks you whether you want to use the **XPath/XQuery Builder** view when you edit long XPath expressions.
- **Show MathFlow recommendation** - displays an information dialog box that recommends using the **MathFlow Editor** to edit MathML equations.
- **Show SFTP certificate warning dialog** - displays a warning dialog box each time the authenticity of the SFTP server host cannot be established.

- **Convert DB Structure to XML Schema** - when tables from a database schema are selected in the **Select Database Table** dialog, after the **Convert DB Structure to XML Schema** is invoked, and another database schema is expanded, a confirmation is needed because the previous selection is discarded. This option controls whether you are always asked about the next action, the other database schema is always expanded without asking you, or it is never expanded.

## Importing / Exporting Global Options

---

The import/export operations are located in the **Options** menu. These operations allow you to load or save global preferences as an XML file. You can use this file to reload saved options both on your computer and on others.

The following actions are available in the **Options** menu:

### Reset Global Options

Restores the preference to the factory defaults, or to the *custom defaults*, if they are defined.

### Import Global Options

Allows you to import a set of *Global Options* from an exported *XML options file*. You can also select a project file (.xpr) to import all the *Global Options* that are set in that project file. After you select a file the **Import Global Options** dialog box is displayed and it informs you that the operation will only override the options that are included in the imported file. You can enable the **Reset all other options to their default values** option to reset all options to the default values before the file is imported.

### Export Global Options

Allows you to export *Global Options* to an XML options file. Some user-specific options that are private are not included. For example, the name of the *Review Author* is not included in the export operation.

Oxygen XML Editor automatically stores your options in an XML options file. Depending on the platform you are using, this file is located in the following directories:

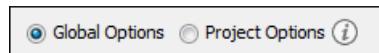
- [user-home-folder]\Application Data\com.oxygenxml for Windows XP
- [user-home-folder]\AppData\Roaming\com.oxygenxml for Windows Vista/7
- [user-home-folder]\Library\Preferences\com.oxygenxml for OS X
- [user-home-folder]\.com.oxygenxml for Linux

The name of the options file of Oxygen XML Editor 17.0 is oxyOptionsSa17.0.xml.

## Project Level Options

---

You are able to set the Oxygen XML Editor options either globally, or to bound them to a specific project by choosing the appropriate setting in the preferences pages:



**Figure 441: Controlling the Storage of the Preferences**

By default, **Global Options** is selected, meaning that the options are stored on your computer, in your user home folder, and are not accessible to other users.

If you select **Project Options**, the preferences are stored in the project file and can be shared with other users. For instance, if your project file, and other files and folders, are saved on a version control system (like SVN, CVS or Source Safe) or a shared folder, your team can use the same options that you have stored in the project file.

**!** **Notice:** Some pages do not have the **Project Options** switch, since the options they host might contain sensitive data (passwords, for example), unsuitable for sharing with other users.

**!** **Note:** If changes have been made to the options in a preferences page and you switch between **Project Options** and **Global Options**, a dialog box will be displayed that allows you to select one of the following:

- **Overwrite** - The existing options from the current preferences page will be overwritten.

- **Preserve** - The existing options from the current preferences page will be preserved.

You may decide that the default schema associations and catalogs must be shared with other team members. To do so, [open the Preferences dialog box](#), go to **XML > XML Catalog**, and select **Project Options**. Now all the options set in the **XML Catalog** page are saved in your current project. At a later time, you can change a preferences panel from being stored in the project file by selecting **Global Options**.

The same mechanism is applied for saving transformation and validation scenarios.

## Reset Global Options

---

To reset all global preferences to their default values, select **Reset Global Options...** from the **Options** menu.

The project level preferences are not changed by this action.

The list of transformation scenarios will be reset to the default scenarios.

## Customizing Default Options

---

Oxygen XML Editor has an extensive set of options that you can configure. When Oxygen XML Editor is installed, these options are set to default values. You can provide a different set of default values for an installation using an *options file*.

### Creating an *options file*

To create an *options file*:

1. You may wish to use a fresh install for this procedure, to make sure that you do not copy personal option settings to the group.
2. Open Oxygen XML Editor and [open the Preferences dialog box](#).
3. Go through the options and set them to the desired defaults. Make sure that you are setting global options, not project options in each page.
4. Close the **Preferences** dialog box.
5. Go to **Options > Export Global Options** and create an XML options file.
6. Go back to the main preferences page and click **Export Global Options** to create an options file.

If you want to control exactly which options page will be stored in the default options file, you can choose to attach them to a *project file (.xpr file extension)* by following these steps:

- In [the Project view](#) create a project or open an already existing one.
- Switch to [Project Options](#) each **Preferences** page which contains settings of interest to you. All explicitly set values will be saved in the project file after you press either of the **OK** or **Apply** buttons.



**Note:** Some pages do not have the **Project Options** switch, since the options they host might contain sensitive data (like passwords, for example), unsuitable for sharing with other users.



**Note:** If you store the options to a *project file (.xpr file extension)* the file extension must be preserved as such (.xpr) when the configuration file is distributed to other users.

### Providing Default Option Values

Use either one of the following ways to configure an Oxygen XML Editor installation to use customized default options from an XML configuration file:

- Set the path to the options file as the value of the `com.oxygenxml.default.options` *system property*.

The path must be specified with an URL or a file path relative to the application installation folder:

```
-Dcom.oxygenxml.default.options=options/default.xml
```

 **Note:** If you are using the *Java Webstart distribution*, edit the `.jnlp` file that launches the application and set the `com.oxygenxml.default.options` parameter using a `property` element, for example:

```
<property name="oxy:com.oxygenxml.default.options"
value="http://host/path/to/default-options.xml"/>
```

- In the `[OXYGEN_DIR]` installation folder, create a folder called `preferences`. Copy the options file in the `[OXYGEN_DIR]/preferences` folder.

 **Note:** The same procedure applies to a *Java Webstart distribution*.

-  **Note:** Make sure that the options configuration file has either the `.xml` extension (for example: `default-options.xml`) or an `.xpr` extension depending on the way in which it was created (from the global options or saved at project level).

## Scenarios Management

---

You can import and export the global transformation and validation scenarios using the following actions:

- To load a set of transformation scenarios from a properties file, go to **Options > Import Global Transformation Scenarios**
- To store a set of transformation scenarios in a properties file, go to **Options > Export Global Transformation Scenarios**
- To load a set of validation scenarios from a properties file, go to **Options > Import Global Validation Scenarios**
- To store all the global (not project-level) validation scenarios in a properties file, go to **Options > Export Global Validation Scenarios**

The **Export Global Transformation Scenarios** and **Export Global Validation Scenarios** options are used to store all the scenarios in a separate properties file. Associations between document URLs and scenarios are also saved in this file. You can load the saved scenarios using the **Import Global Transformation Scenarios** and **Import Global Validation Scenarios** actions. To distinguish the existing scenarios and the imported ones, the names of the imported scenarios contain the word **import**.

## Editor Variables

---

An editor variable is a shorthand notation for context-dependent information, such as a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example, the input URL of a transformation, the output file path of a transformation, or the command line of an external tool) to make a command or a parameter generic and re-usable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Editor commands of external engines or other external tools, in transformation scenarios, Author operations, and in validation scenarios:

- `${oxygenHome}` - Oxygen XML Editor installation folder as URL.
- `${oxygenInstallDir}` - Oxygen XML Editor installation folder as file path.
- `${framework}` - The path (as URL) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- `${framework(fr_name)}` - The path (as URL) of the `fr_name` framework.
- `${frameworkDir(fr_name)}` - The path (as file path) of the `fr_name` framework.



**Note:** Because multiple frameworks might have the same name (although it is not recommended), for both  `${framework(fr_name)}`  and  `${frameworkDir(fr_name)}`  editor variables Oxygen XML Editor employs the following algorithm when searching for a given framework name:

- all frameworks are sorted, from high to low, according to their **Priority** setting from the [Document Type Association preferences page](#). Only frameworks that have the **Enabled** checkbox set are taken into account.
- next, if the two or more frameworks have the same name and priority, a further sorting based on the **Storage** setting is made, in the exact following order:
  - frameworks stored in the internal Oxygen XML Editor options
  - additional frameworks added in the [Locations preferences page](#)
  - frameworks installed using the add-ons support
  - frameworks found in the [main frameworks location](#) (**Default** or **Custom**)

- `${frameworks}`  - The path (as URL) of the  `[ OXYGEN_DIR ] /frameworks`  directory.
- `${frameworkDir}`  - The path (as file path) of the current framework, as part of the  `[ OXYGEN_DIR ] /frameworks`  directory.
- `${frameworksDir}`  - The path (as file path) of the  `[ OXYGEN_DIR ] /frameworks`  directory.
- `${home}`  - The path (as URL) of the user home folder.
- `${homeDir}`  - The path (as file path) of the user home folder.
- `${pdu}`  - Current project folder as URL. Usually the current folder selected in the Project View.
- `${pd}`  - Current project folder as file path. Usually the current folder selected in the Project View.
- `${pn}`  - Current project name.
- `${cfdu}`  - Current file folder as URL, that is the path of the current edited document up to the name of the parent folder, represented as a URL.
- `${cfdu}`  - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- `${cfn}`  - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- `${cfne}`  - Current file name with extension. The current file is the one currently opened and selected.
- `${cf}`  - Current file as file path, that is the absolute file path of the current edited document.
- `${af}`  - The local file path of the ZIP archive that includes the current edited document.
- `${afu}`  - The URL path of the ZIP archive that includes the current edited document.
- `${afd}`  - The local directory path of the ZIP archive that includes the current edited document.
- `${afdu}`  - The URL path of the directory of the ZIP archive that includes the current edited document.
- `${afn}`  - The file name (without parent directory and without file extension) of the zip archive that includes the current edited file.
- `${afne}`  - The file name (with file extension, for example  `.zip`  or  `.epub` , but without parent directory) of the zip archive that includes the current edited file.
- `${currentFileURL}`  - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- `${ps}`  - Path separator, that is the separator which can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- `${timeStamp}`  - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- `${caret}`  - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `${selection}`  - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- `${id}`  - Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.

- `${uuid}`  - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java [UUID](#) class.
- `${env(VAR_NAME)}`  - Value of the `VAR_NAME` environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the  `${system(var.name)}`  editor variable.
- `${system(var.name)}`  - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the  `${env(VAR_NAME)}`  editor variable instead.
- `${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}`  - To prompt for values at runtime, use the  `${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')}`  editor variable. You can set the following parameters:
  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - `type` - Optional parameter, with one of the following values:

Parameter	
url	<p><b>Format:</b> <code> \${ask('message', url, 'default_value')} </code></p> <p><b>Description:</b> Input is considered a URL. Oxygen XML Editor checks that the provided URL is valid.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Input URL', url)} </code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL.</li> <li>• <code> \${ask('Input URL', url, 'http://www.example.com')} </code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL. The input field displays the default value <code>http://www.example.com</code>.</li> </ul>
password	<p><b>Format:</b> <code> \${ask('message', password, 'default')} </code></p> <p><b>Description:</b> The input is hidden with bullet characters.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Input password', password)} </code> - The displayed dialog box has the name <code>'Input password'</code> and the input is hidden with bullet symbols.</li> <li>• <code> \${ask('Input password', password, 'abcd')} </code> - The displayed dialog box has the name <code>'Input password'</code> and the input hidden with bullet symbols. The input field already contains the default <code>abcd</code> value.</li> </ul>
generic	<p><b>Format:</b> <code> \${ask('message', generic, 'default')} </code></p> <p><b>Description:</b> The input is considered to be generic text that requires no special handling.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Hello world!')} </code> - The dialog box has a <code>Hello world!</code> message displayed.</li> <li>• <code> \${ask('Hello world!', generic, 'Hello again!')} </code> - The dialog box has a <code>Hello world!</code> message displayed and the value displayed in the input box is <code>'Hello again!'</code>.</li> </ul>
relative_url	<p><b>Format:</b> <code> \${ask('message', relative_url, 'default')} </code></p> <p><b>Description:</b> Input is considered a URL. Oxygen XML Editor tries to make the URL relative to that of the document you are editing.</p> <p> <b>Note:</b> If the <code>\$ask</code> editor variable is expanded in content that is not yet saved (such as an <i>untitled</i> file, whose path cannot be determined), then Oxygen XML Editor will transform it into an absolute URL.</p>

Parameter	
	<p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('File location', relative_url, 'C:/example.txt') } - The dialog box has the name 'File location'. The URL inserted in the input box is made relative to the current edited document location.</code></li> </ul>
combobox	<p><b>Format:</b> <code> \${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default') }</code></p> <p><b>Description:</b> Displays a dialog box that offers a drop-down list. The drop-down list is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated value (<code>real_value</code>).</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx') } - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems. The associated value will be returned based upon your selection.</code></li> </ul> <p> <b>Note:</b> In this example Mac OS X is the default selected value and if selected it would return <code>osx</code> for the output.</p>
editable_combobox	<p><b>Format:</b> <code> \${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default') }</code></p> <p><b>Description:</b> Displays a dialog box that offers a drop-down list with editable elements. The drop-down list is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated real value (<code>real_value</code>) or the value inserted when you edit a list entry.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx') } - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.</code></li> </ul>
radio	<p><b>Format:</b> <code> \${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default') }</code></p> <p><b>Description:</b> Displays a dialog box that offers a series of radio buttons. Each radio button displays a '<code>rendered_value</code>' and will return an associated <code>real_value</code>.</p> <p><b>Example:</b></p> <ul style="list-style-type: none"> <li>• <code> \${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx') } - The dialog box has the name 'Operating System'. The radio button group allows you to choose between the three operating systems.</code></li> </ul> <p> <b>Note:</b> In this example Mac OS X is the default selected value and if selected it would return <code>osx</code> for the output.</p>

- '`default-value`' - optional parameter. Provides a default value.
- `$(date(pattern))` - Current date. The allowed patterns are equivalent to the ones in the [Java SimpleDateFormat class](#).  
**Example:** `YYYY-MM-dd`;



**Note:** This editor variable supports both the `xs:date` and `xs:datetime` parameters. For details about `xs:date`, go to <http://www.w3.org/TR/xmlschema-2/#date>. For details about `xs:datetime`, go to <http://www.w3.org/TR/xmlschema-2/#dateTime>.

- `$(dbgXML)` - The local file path to the XML document which is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- `$(dbgXSL)` - The local file path to the XSL/XQuery document which is current selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- `$(tsf)` - The transformation result file path. If the current opened file has an associated scenario which specifies a transformation output file, this variable expands to it.
- `$(dsu)` - The path of the detected schema as an URL for the current validated XML document.
- `$(ds)` - The path of the detected schema as a local file path for the current validated XML document.
- `$(cp)` - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- `$(tp)` - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor / Print** Preferences page.
- `$(xpath_eval(expression))` - Evaluates an XPath 3.0 expression. Depending on the context, the expression can be:
  - *static*, when executed in a non-XML context. For example, you can use such static expressions to perform string operations on other editor variables for composing the name of the output file in a transformation scenario's **Output** tab.

**Example:**

```
 ${xpath_eval(upper-case(substring('${cfn}', 1, 4)))}
```

- *dynamic*, when executed in an XML context. For example, you can use such dynamic expression in a code template or as a value of an author operation's parameter.

**Example:**

```
 ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
```

- `$(i18n(key))` - Editor variable used only at document type/framework level to allow translating names and descriptions of Author actions in multiple actions. For more details see the [Localizing Frameworks](#) on page 548 section.

## Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of an external tool or of a custom validator, the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

You can configure the custom editor variables in the [Custom Editor Variables preferences page](#).

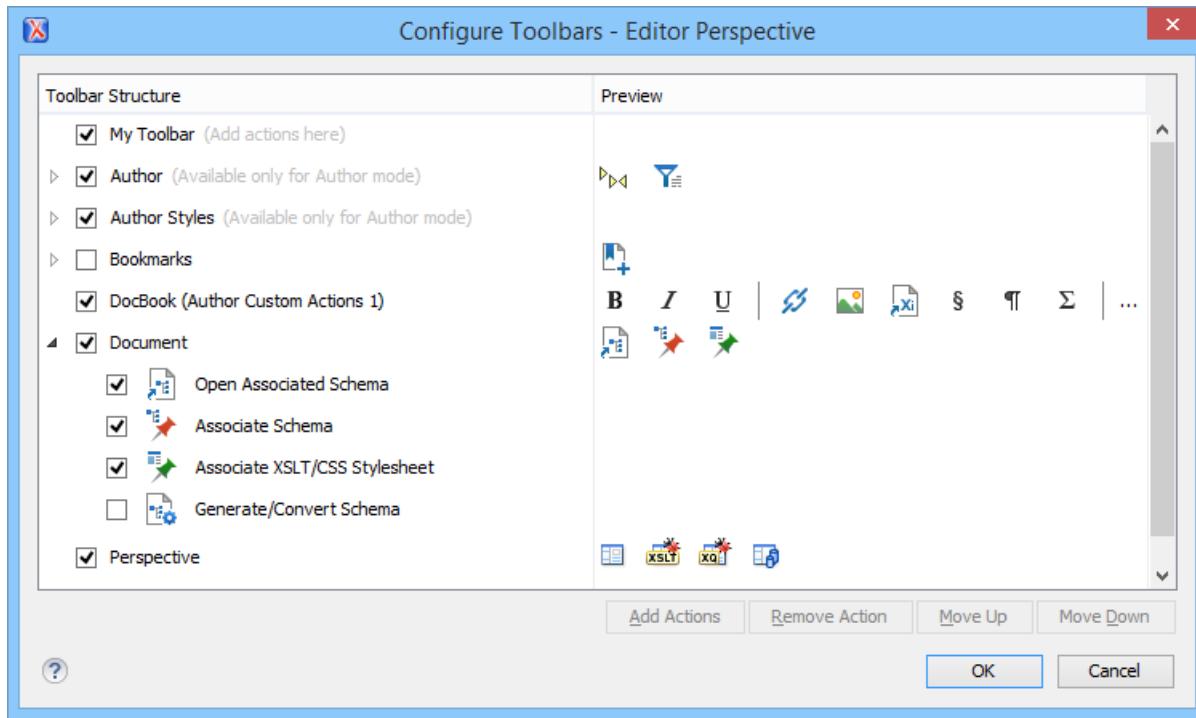
## Configure Toolbars

You can configure the toolbars in Oxygen XML Editor to personalize the interface for your specific needs. You can choose which toolbars to show or hide in the current editor mode (**Text**, **Author**, **Design**, or **Grid**) and in the current

perspective (**Editor**, **XSLT Debugger**, **XQuery Debugger**, or **Database**). You can also choose which actions to display in each toolbar, add actions to toolbars, and customize the layout of the toolbars.

To configure the toolbars, open the **Configure Toolbars** dialog box by doing one of the following:

- Right-click on any toolbar and select **Configure toolbars ...**.
- Select **Configure toolbars ...** from the **Window** menu.



**Figure 442: Configure Toolbars Dialog Box**

The **Configure Toolbars** dialog box provides the following actions:

#### Show or Hide Toolbars

You can choose whether to show or hide a toolbar by using the checkbox next to the toolbar name. This checkbox is only available for toolbars that are available for the current perspective and editing mode.

#### Show or Hide Actions in a Toolbar

To show or hide actions in a toolbar, expand it by clicking on the arrow next to the toolbar name, then use the checkbox to select or deselect the appropriate actions. The toolbar configuration changes in the **Preview** column according to your changes.

#### Add Actions to a Toolbar

Use the **Add Actions** button to open the **Add Actions** dialog box that displays all the actions that can be added to any of the toolbars, with the exception of those that are contributed from frameworks (document type associations) or 3rd party plugins.

#### Remove Actions from a Toolbar

You can remove actions that you have previously added to toolbars by using the **Remove Action** button.

#### Move Actions in a Toolbar

Use the **Move Up** and **Move Down** actions to change the order of the actions in a toolbar.

The **Configure Toolbars** dialog box also provides a variety of other ways to customize the layout in Oxygen XML Editor.

## Customize My Toolbar

You can customize the **My Toolbar** to include your most commonly used actions. By default, this toolbar is listed first. Also, it is hidden until you add actions to it and you can easily hide it with the **Hide "My Toolbar" Toolbar** action that is available when you right-click anywhere in the toolbar area.

## Drop-down List Actions

Composite actions that are usually displayed as a drop-down list can only be selected in one toolbar at a time. These actions are displayed in the **Configure Toolbars** dialog box with the name in brackets.



## Configure External Tools Action

There is a **Configure external tools** composite action that appears in the toolbar called **Tools**. It is a drop-down list that contains any external tools that are configured in the **External Tools** preferences page.

**Note:** If no external tools are configured, this drop-down list is not shown in the toolbar.

Additional actions are available from the **Window** menu or contextual menu when invoked from a toolbar that allows you to further customize your layout. These actions include:

### Reset Toolbars

To reset the layout of toolbars to the default setting, select the **Reset Toolbars** action from the contextual menu or **Window** menu.

### Reset Layout

To reset the entire layout (including toolbars, editing modes, views, etc.) to the default setting, select **Reset Layout** from the contextual menu or **Window** menu.

### Export Layout

You can use the **Export Layout** action that is available in the **Window** menu to export the entire layout of the application to share it with other users.

### Hide Toolbars

You can use the **Hide Toolbar** action from the contextual menu to easily hide a displayed toolbar. When you right-click on a toolbar it will be highlighted to show you which actions are included in that toolbar.

## Custom System Properties

---

A number of Java system properties can be set in the application to influence its behavior:

**Table 18: Custom System Properties**

Property	Allowed values	Purpose
<code>com.oxygenxml.disable.http.protocol.handlers</code>	true/false (false by default)	By default Oxygen XML Editor uses for HTTP(S) connections the open source Apache HTTP Client software. If you disable this feature, the default Java SUN protocols HTTP(S) will be used instead. You will also lose <b>WebDAV</b> support and probably other related features.
<code>com.oxygenxml.default.options</code>	An URL-like relative or absolute path.	Provides the path to an XML file containing default application options. Read this topic for more details: <a href="#">Customizing Default Options</a> on page 1085.

Property	Allowed values	Purpose
<code>com.oxygenxml.customOptionsDir</code>	A file system absolute path pointing to a folder.	Sets a different folder which will be used by the application to load and save preference files. The default place where the options are saved varies according to the operating system: <a href="#">Importing / Exporting Global Options</a> on page 1084
<code>com.oxygenxml.ApplicationDataFolder</code>	A file system absolute path pointing to a folder.	When the application runs on Windows you can set this property to it to change the place where the application considers that the <code>APPDATA</code> folder is located.
<code>com.oxygenxml.editor.frameworks.url</code>	An URL-like absolute path.	Changes the folder where the application considers that the main frameworks/document types are installed. It has the same effect as changing the custom frameworks directory value in the <a href="#">Locations Preferences</a> on page 1005 Preferences page.
<code>com.oxygenxml.MultipleInstances</code>	true/false	If <b>true</b> allows multiple instances of the application to be started. By default it is <b>true</b> on Linux and <b>false</b> on Windows.
<code>com.oxygenxml.xep.location</code>	A file system absolute path pointing to a folder.	Points to a folder where RenderX XEP is installed. Has the same effect as configuring XEP in the <a href="#">FO Processors Preferences</a> on page 1057 Preferences page.
<code>com.oxygenxml.additional.classpath</code>	A list of JAR-like resources separated by the platform file separator.	An additional list of libraries to be used in the application's internal class loader in addition to the libraries specified in the <code>lib</code> folder.
<code>com.oxygenxml.user.home</code>	A file system absolute path pointing to a folder.	Overwrites the user home which was implicitly detected for the application. Used only when the application is running on Windows.
<code>com.oxygenxml.use.late.delegation.for.authorextensions</code>	true/false (true by default)	All Java extensions in a document type configuration are instantiated in a separate class loader. When <b>true</b> , the JAR libraries used in a certain document type configuration will have priority to resolve classes before delegating to the parent class loader. This is the default behavior. When <b>false</b> the parent class loader will take precedence.
<code>com.oxygenxml.stack.size.validation.threads</code>	The number of bytes used to validation threads.	Some parts of the application (validation, content completion) which use the Relax NG parser require sometimes a larger Thread stack size in order to parse complex schemas. The default value is $(5 * 1024 * 1024)$ which should be more than enough.
<code>com.oxygenxml.jing.skip.validation.xhtml.data.attrs</code>	true/false (true by default)	By default the Relax NG validation was configured to skip validation for XHTML attributes starting with "data-" which should be skipped from validation according to the XHTML 5 specification.

Property	Allowed values	Purpose
<code>com.oxygenxml.report.problems.url</code>	User defined URL	The URL where a problem reported through the <b>Report Problem</b> dialog box is sent. The report is sent in XML format using the <code>report</code> parameter with the POST HTTP method.

## Localizing the User Interface

You can select the language of the Oxygen XML Editor user interface. Oxygen XML Editor ships with the following languages: English, French, German, Japanese, and Dutch. To change the interface language, go to **Options > Preferences > Global** preferences page, then choose the appropriate language from the **Language** drop-down list.

You can also localize the interface in a different language by [creating an interface localization file](#).

### Creating an Interface Localization File

You can change the language of the Oxygen XML Editor user interface by creating an interface localization file. The example in this procedure is based on the Spanish language, and a standard Oxygen XML Editor Windows distribution.

1. Identify the code for the new language you wish to translate the interface. It is composed from a language code (two or three lowercase letters that conform to the [ISO 639 standard](#)), followed by an underscore character, and a region code (two or three uppercase letters that conform to the [ISO 3166 standard](#)).
2. Write an email to the Oxygen XML Editor support team and ask them to send you the `translation.xml` sample file.
3. Open `translation.xml` in Oxygen XML Editor. The file contains all the interface messages that can be translated and is updated at every new release with the latest additions. Here is a sample of its content:

```
<translation>
 <languageList>
 <language description="English" lang="en_US"/>
 </languageList>
 <key value="New">
 <comment>The File/New action. Creates a new document.</comment>
 <val lang="en_US">New</val>
 </key>
 <key value="New_folder">
 <comment>Creates a folder in the Project View.</comment>
 <val lang="en_US">New Folder</val>
 </key>
 ...
</translation>
```

4. Update the `language` element to reflect the new language. Set the `description` attribute to Spanish and the `lang` attribute to `es_ES`.
5. For each `key` element translate the `comment` (optional) and `val` elements. Set the `lang` attribute to `es_ES`.



**Note:** After you are finished, the file should look like:

```
<translation>
 <languageList>
 <language description="Español" lang="es_ES"/>
 </languageList>
 <key value="New">
 <comment>El Archivo / Nueva acción. Crea un nuevo documento.</comment>
 <val lang="es_ES">Nuevo</val>
 </key>
 <key value="New_folder">
 <comment>Crea una carpeta en la vista del proyecto.</comment>
 <val lang="es_ES">Nueva carpeta</val>
 </key>
 ...
</translation>
```

6. [Open the Preferences dialog box](#) and go to **Global** and enable the **Other language** option. Browse for the `translation.xml` file.

7. Restart the application.

## Setting a Java Virtual Machine Parameter in the Launcher Configuration File / Start-up Script

---

There are two ways you can set new Java Virtual Machine parameters:

- *Setting parameters for the Oxygen XML Editor launchers*
- *Setting parameters in the command line scripts*

### Setting Parameters for the Application Launchers

#### Increasing the amount of memory that Oxygen XML Editor uses on Windows

To increase the memory available to Oxygen XML Editor on Windows:

- Navigate to the installation directory of Oxygen XML Editor.
- Locate the -Xmx parameter in the `oxygen17.0.vmoptions` file;

 **Note:** For 32-bit Windows modify the parameter to `-Xmx1024m` or larger, but not over `-Xmx1200m`. Make sure you do not exceed your physical RAM. For 64-bit Windows modify the parameter to a larger value (for example `-Xmx2048m`). We recommended you to not use more than half of your existing physical RAM.

Restart Oxygen XML Editor. Go to **Help > About** and verify the amount of memory that is actually available (see the last row in the **About** dialog). In case Oxygen XML Editor does not start and you receive an error message saying that it could not start the JVM, decrease the `-Xmx` parameter and try again.

For Windows Vista/7, copy the `oxygen17.0.vmoptions` to your desktop (or to any other folder with write access), modify it there, then copy it back to the Oxygen XML Editor installation folder.

 **Note:** The parameters from the `.vmoptions` file are used when you start Oxygen XML Editor with the *oxygen* launcher (or with the desktop shortcut). In case you use the command line script `oxygen.bat/oxygen.sh`, modify the `-Xmx` parameter in the script file.

#### Increasing the amount of memory that Oxygen XML Editor uses on Mac OS X

To increase the memory available to Oxygen XML Editor on Mac OS X:

- **Ctrl Click (Command Click on OS X)** (or right click) the Oxygen XML Editor icon in **Finder**.
- From the pop-up menu, select **Show Package Contents**.
- Navigate to the **Contents** directory and open for editing the `Info.plist` file.

 **Note:** You can open this file either with the **Property List Editor**, or the **TextEdit**.

- Look for the **VMOptions** key and adjust the `-Xmx` parameter to a larger value (for example `-Xmx1500m`)

 **Note:** For a Mac kit bundled with Java 7, look for the **VMOptionArray** key and add the `-Xmx` parameter in a new string element from the `array` element. For example, for 1500 MB use the following:

```
<string>-Xmx1500m</string>
```

 **Tip:** Try not to use more than half of your existing physical RAM if possible.

## Setting a system property

To set a system property, you have to provide a parameter of the following form:

```
-Dproperty.name=value
```

You can also set a system property through a parameter prefixed with **-Doxy** in the command line used to start the application:

```
oxygen17.0.exe "-Doxyproperty.name=value"
```

All system properties are displayed in the **System properties** tab of the **About** dialog.

To view the list of Oxygen XML Editor system properties, go to [Custom System Properties](#) on page 1092.

## Disabling DPI Scaling

Some users may prefer the look of smaller icons in a HiDPI display. To achieve this, display scaling needs to be disabled for high DPI settings. To disable the DPI scaling, set the following property in `.vmoptions` (or in the `.bat` script):

```
sun.java2d.dpiaware=false
```

## Setting Parameters in the Command Line Scripts

If you start Oxygen XML Editor with the `oxygen.bat` script, you have to add or modify the parameter to the `java` command at the end of the script.

For example, to set the maximum amount of Java memory to 600 MB on **Windows**, modify the **-Xmx** parameter like this:

```
java -Xmx600m -Dsun.java2d.noddraw=true ...
```

on **Mac OS X** the `java` command should look like:

```
java "-Xdock:name=Oxygen"\
-Dcom.oxygenxml.editor.plugins.dir="$OXYGEN_HOME/plugins"\
-Xmx600m\
...
```

and on **Linux** the `Java` command should look like:

```
java -Xmx600m\
"-Dcom.oxygenxml.editor.plugins.dir=$OXYGEN_HOME/plugins"\
```

---

# Chapter

# 22

---

## Common Problems

---

**Topics:**

- *Performance Problems*
- *Common Problems and Solutions*

This section lists the most commonly found problems and their solutions.

## Performance Problems

---

This section contains the solutions for some common problems that may appear when running Oxygen XML Editor.

### Large Documents

When started from the icon created on the Start menu or the Desktop on Windows or from the shortcut created on the Linux desktop the maximum memory available to Oxygen XML Editor is set by default to 40% of the amount of physical RAM but not more than 700 MB. When started from the command line scripts the maximum memory is 256 MB. If large documents are edited in Oxygen XML Editor and you see that performance slows down considerably after some time then a possible cause is that the application needs more memory in order to run properly. You can increase the maximum amount of memory available to Oxygen XML Editor by [setting the -Xmx parameter in a configuration file](#) specific to the platform that runs the application.



#### Attention:

The maximum amount of memory should not be equal to the physical amount of memory available on the machine because in that case the operating system and other applications will have no memory available.

When installed on a multi-user environment such as Windows Terminal Server or Unix/Linux, to each instance of Oxygen XML Editor will be allocated the amount stipulated in the memory value. To avoid degrading the general performance of the host system, please ensure that the amount of memory available is optimally apportioned for each of the expected instances.

### External Processes

The [\*Memory available to the built-in FOP\*](#) option controls the amount of memory allocated to generate PDF output with the built-in Apache FOP processor. If Oxygen XML Editor throws an *Out Of Memory* error, [open the Preferences dialog box](#), go to **XML > XSLT-FO-XQuery > FO Processors**, and increase the value of the [\*Memory available to the built-in FOP\*](#) option.

For external XSL-FO processors, XSLT processors, and external tools, the maximum value of the allocated memory is set in the command line of the tool using the -Xmx parameter set to the Java virtual machine.

### Display Problems on Linux or Solaris

Display problems like screen freeze or momentary menu pop-ups during mouse movements over screen on Linux or Solaris can be solved by [adding the startup parameter](#) -Dsun.java2d.pmoffscreen=false.

## Common Problems and Solutions

---

This chapter presents common problems that may appear when running the application and the solutions for these problems.

### XML Document Takes a Long Time to Open

If Oxygen XML Editor takes a long time to open a document, check the following:

It takes longer to open an XML document if the whole content of your document is on a single line or if the document size is very large.

If the content is on a single line, you can speed up loading by enabling the **Format and indent the document on open** option. To do so, [open the Preferences dialog box](#) and go to **Editor > Format > Format and indent the document on open**.

If the document is very large (above 30 MB) make sure that the value of [\*Optimize loading in the Text edit mode for files over\*](#) is greater than the size of your document.

If that fails and you get an Out Of Memory error (**OutOfMemoryError**) you can [increase the memory available to Oxygen XML Editor](#).

## Oxygen XML Editor Takes Several Minutes to Start on Mac

If Oxygen XML Editor takes several minutes to start, the Java framework installed on the Mac may have a problem. One solution for this is to update Java to the latest version: go to **Apple symbol > Software Update**. After it finishes to check for updates, click **Show Details**, select the Java Update (if one is available) and click **Install**. If no Java updates are available, reset the Java preferences to their defaults. Start **Applications > Utilities > Java Preferences** and click **Restore Defaults**.

## Out Of Memory Error When I Open Large Documents

I am trying to open a file larger than 100 MB to edit it in Oxygen XML Editor, but it keeps telling me it runs out of memory (**OutOfMemoryError**). What can I do?

You should make sure that the minimum limit of document size that enables the support for editing large documents (the value of [the option Optimize loading in the Text edit mode for files over](#)) is less than the size of your document. That will enable the optimized support for large documents. If that fails and you still get an Out Of Memory error you should [increase the memory available to Oxygen XML Editor](#).

Other tips:

- Make sure that you close other files before opening the large file.
- You can set the default editing mode [in the Preferences dialog](#). The *Text editing mode* uses less memory than other editing modes.
- If the file is too large for the editor to handle, you can [open it in for viewing in Large File Viewer](#).

## Special Characters Are Replaced With a Square in Editor

My file was created with other application and it contains special characters like é, ©, ®, etc. Why does Oxygen XML Editor display a square for these characters when I open the file in Oxygen XML Editor?

You must set a font able to render the special characters in the [Font preferences](#). If it is a text file you must set also [the encoding used for non XML files](#). If you want to set a font which is installed on your computer but that font is not accessible in the **Font** preferences that means the Java virtual machine is not able to load the system fonts, probably because it is not a True Type font. It is a problem of the Java virtual machine and a possible solution is to copy the font file in the `[JVM_DIR]/lib/fonts` folder. `[JVM_DIR]` is the value of the property `java.home` which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help > About**.

## XSLT Debugger Is Very Slow

When I run a transformation in the **XSLT Debugger** perspective it is very slow. Can I increase the speed?

If the transformation produces HTML or XHTML output you should [disable rendering of output in the XHTML output view](#) during the transformation process. To view the XHTML output result do one of the following:

- Run the transformation in the **Editor** perspective and make sure the [Open in Browser/System Application option](#) is enabled.
- Run the transformation in the **XSLT Debugger** perspective, save the text output area to a file, and use a browser application for viewing it (for example Firefox or Internet Explorer).

## The Scroll Function of my Notebook's Trackpad is Not Working

I got a new notebook (Lenovo Thinkpad™ with Windows) and noticed that the scroll function of my trackpad is not working in Oxygen XML Editor.

It is a problem of the Synaptics™ trackpads which can be fixed by adding the following lines to the `C:\Program Files\Synaptics\SynTP\TP4table.dat` file:

```
* , * , oxygen17.0.exe,* , * , * , WheelStd,1,9
* , * , oxygenAuthor17.0.exe,* , * , * , WheelStd,1,9
* , * , oxygenDeveloper17.0.exe,* , * , * , WheelStd,1,9
* , * , syncroSVNClient.exe,* , * , * , WheelStd,1,9
* , * , diffDirs.exe,* , * , * , WheelStd,1,9
* , * , diffFiles.exe,* , * , * , WheelStd,1,9
```

## NullPointerException at Startup on Windows XP

When I start Oxygen XML Editor on Windows XP I get the following error. What can I do?

```
Cannot start Oxygen XML Editor.
Due to:java.lang.NullPointerException
java.lang.NullPointerException
at com.sun.java.swing.plaf.windows.XPStyle.getString(Unknown Source)
at com.sun.java.swing.plaf.windows.XPStyle.getString(Unknown Source)
at com.sun.java.swing.plaf.windows.XPStyle.getDimension(Unknown Source)
at com.sun.java.swing.plaf.windows.WindowsProgressBarUI.
getPreferredInnerHorizontal(Unknown Source)
```

The error is caused by a bug in the Java runtime from Sun Microsystems. You can avoid it by setting the Java system property `com.oxygenxml.no.xp.theme` to the value `true` in the startup script, that is [adding the startup parameter -Dcom.oxygenxml.no.xp.theme=true](#). If you start Oxygen XML Editor with the `oxygen.bat` script just add the parameter `-Dcom.oxygenxml.no.xp.theme=true` to the `java` command in the script. If you start Oxygen XML Editor from the Start menu shortcut you have to add the same parameter on a new line in the file `[oxygen-install-folder]\oxygen17.0.vmoptions`.

## Crash at Startup on Windows with an Error Message About a File nvoglv32.dll

I try to start Oxygen XML Editor on Windows but it crashed with an error message about “Fault Module Name: `nvoglv32.dll`”. What is the problem?

It is an OpenGL driver issue that can be avoided by creating an empty file called `opengl32.dll` in the Oxygen XML Editor install folder (if you start Oxygen XML Editor with the shortcut created by the installer on the Start menu or on Desktop) or in the subfolder `bin` of the home folder of the Java virtual machine that runs Oxygen XML Editor (if you start Oxygen XML Editor with the `oxygen.bat` script). The home folder of the Java virtual machine that runs Oxygen XML Editor is the value of the `java.home` property that is available in the **System properties** tab of the **About** dialog box (opened from the **Help > About** menu).

## Oxygen XML Editor Crashed on My Mac OS X Computer

Oxygen XML Editor crashed the Apple Java virtual machine/Oxygen XML Editor could not start up on my OS X computer due to a JVM crash. What can I do?

Usually it is an incompatibility between the Apple JVM and a native library of the host system. More details are available in the crash log file generated by OS X and reported in the crash error message.

## Wrong Highlights of Matched Words in a Search in User Manual

When I do a keyword search in the User Manual that comes with the Oxygen XML Editor application the search highlights the wrong word in the text. Sometimes the highlighted word is several words after the matched word. What can I do to get correct highlights?

This does not happen when Oxygen XML Editor runs with a built-in Java virtual machine, that is a JVM that was installed by Oxygen XML Editor in a subfolder of the installation folder, for example on Windows and Linux when installing Oxygen XML Editor with the installation wizard specific for that platform. When Oxygen XML Editor runs from an All Platforms installation it uses whatever JVM was found on the host system which may be incompatible with the JavaHelp indexer used for creating the built-in User Manual. Such a JVM may offset the highlight of the matched word with several characters, usually to the right of the matched word. In order to see the highlight exactly on the matched word it is recommended to install the application with the specific installation wizard for your platform (available only for Windows and Linux).

## Keyboard Shortcuts Do Not Work

The keyboard shortcuts listed in the [Menu Shortcut Keys preferences](#) do not work. What can I do?

Usually this happens when a special keyboard layout is set in the operating system which generates other characters than the usual ones for the keys of a standard keyboard. For example if you set the extended Greek layout for your keyboard you should return to the default Greek layout or to the English one. Otherwise the Java virtual machine that runs the application will receive other key codes than the usual ones for a standard keyboard.

## After Installing Oxygen XML Editor I Cannot Open XML Files in Internet Explorer Anymore

Before installing Oxygen XML Editor I had no problems opening XML files in Internet Explorer. Now when I try to open an XML file in Internet Explorer it opens the file in Oxygen XML Editor. How can I load XML files in Internet Explorer again?

XML files are opened in Oxygen XML Editor because Internet Explorer uses the Windows system file associations for opening files and you associated XML files with Oxygen XML Editor in the installer panel called **File Associations** therefore Internet Explorer opens XML files with the associated Windows application.

By default the association with XML files is disabled in the Oxygen XML Editor installer panel called **File Associations**. When you enabled it the installer displayed a warning message about the effect that you experience right now.

For opening XML files in Internet Explorer again you have to set Internet Explorer as the default system application for XML files, for example by right-clicking on an XML file in Windows Explorer, selecting **Open With > Choose Program**, selecting IE in the list of applications and selecting the checkbox **Always use the selected program**. Also you have to run the following command from a command line:

```
wscript revert.vbs
```

where revert.vbs is a text file with the following content:

```
function revert()
 Set objShell = CreateObject("WScript.Shell")
 objShell.RegWrite "HKCR\.xml\", "xmlfile", "REG_SZ"
 objShell.RegWrite "HKCR\.xml\Content Type", "text/xml", "REG_SZ"
end function

revert()
```

## I Cannot Associate Oxygen XML Editor With a File Type on My Windows Computer

I cannot associate the Oxygen XML Editor application with a file type on my Windows computer by right clicking on a file in Windows Explorer, selecting **Open With > Choose Program** and browsing to the file oxygen17.0.exe. When I select the file oxygen17.0.exe in the Windows file browser dialog box, the Oxygen XML Editor application is not added to the list of applications in the **Open With** dialog box. What can I do?

The problem is due to some garbage Windows registry entries remained from versions of Oxygen XML Editor older than version 9.0. Please uninstall all your installed versions of Oxygen XML Editor and run a registry cleaner application for cleaning these older entries. After that just reinstall your current version of Oxygen XML Editor and try again to create the file association.

## The Files Are Opened in Split Panels When I Restart Oxygen XML Editor

When I close the Oxygen XML Editor application with multiple files open and then restart it, every file opens in a split panel of the editing area instead of a tab sharing with the other opened files the same editing area which organizes the editors in a tabbed pane. I want to have the files arranged as a tabbed pane as they used to be arranged before closing the Oxygen XML Editor application.

This happens sometimes when several files are opened automatically on startup. It is a problem of the JIDE docking views library used in Oxygen XML Editor for docking and floating views. The workaround is to use the **Reset Layout** action from the **Window** menu.

## Grey Window on Linux With the Compiz / Beryl Window Manager

I try to run Oxygen XML Editor on Linux with the Compiz / Beryl window manager but I get only a grey window which does not respond to user actions. Sometimes the Oxygen XML Editor window responds to user actions but after opening and closing an Oxygen XML Editor dialog or after resizing the Oxygen XML Editor window or a view of the Oxygen XML Editor window the content of this window becomes grey and it does not respond to user actions. What is wrong?

Sun Microsystems' Java virtual machine [does not support the Compiz window manager and the Beryl one very well](#). It is expected that better support for Compiz / Beryl will be added in future versions of their Java virtual machine. You should turn off the special effects of the Compiz / Beryl window manager before starting the Oxygen XML Editor application or switch to other window manager.

## Drag and Drop Without Initial Selection Does Not Work

When I try to drag with the mouse an unselected file from the **Project** view or the **DITA Maps Manager** view, the drag never starts, it only selects the resource. I need to drag the resource again after it becomes selected. As a result any drag and drop without initial selection becomes a two step operation. How can I fix this?

This is [a bug](#) present in JVM versions prior to 1.5.0\_09. This issue is fixed in 1.5.0\_09 and newer versions. See [the installation instructions](#) for setting a specific JVM version for running the Oxygen XML Editor application.

## Set Specific JVM Version on Mac OS X

How do I configure Oxygen XML Editor to run with the version X of the Apple Java virtual machine on my Mac OS X computer?

Oxygen XML Editor uses the first JVM from the list of preferred JVM versions set on your Mac computer that has a version number 1.6.0 or higher. You can move your desired JVM version up in the preferred list by dragging it with the mouse on a higher position in the list of JVMs available in the **Java Preferences** panel that is opened from **Applications > Utilities > Java > Java Preferences**.

## Segmentation Fault Error on Mac OS X

On my Mac OS X machine the application gives a *Segmentation fault* error when I double-click on the application icon. Sometimes it gives no error but it does not start. What is the problem?

Please make sure you have the latest Java update from the Apple website installed on your Mac OS X computer. If installing the latest Java update doesn't solve the problem please copy the file `JavaApplicationStub` from the `/System/Frameworks/JavaVM.framework` folder to the `Oxygen.app/Contents/MacOS` folder. For browsing the `.app` folder you have to [\*\*\(CMD+click\)\*\*](#) on the Oxygen XML Editor icon and select **Show Package Contents**.

## Damaged File Associations on OS X

After upgrading OS X and Oxygen XML Editor, it is no longer associated to the appropriate file types (such as XML, XSL, XSD, etc.) How can I create the file associations again?

The upgrade damaged the file associations in the LaunchService Database on your OS X machine. Please rebuild the LaunchService Database with the following procedure. This will reset all file associations and will rescan the entire file system searching for applications that declare file associations and collecting them in a database used by Finder.

1. Find all the Oxygen XML Editor installations on your hard drive.
2. Delete them by dragging them to the Trash.
3. Clear the Trash.
4. Unpack the Oxygen XML Editor installation kit on your desktop.
5. Copy the contents of the archive into the folder `/ Applications / Oxygen`.
6. Run the following command in a Terminal:

```
/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/LaunchServices.framework/Versions/A/Support/lsregister
-kill -r -domain local -domain system -domain user
```

7. Restart Finder with the following command:

```
killall Finder
```

8. Create a XML or XSD file on your desktop.  
It should have the Oxygen XML Editor icon.
9. Double-click the file.
10. Accept the confirmation.

When you start Oxygen XML Editor the file associations should work correctly.

## I Cannot Connect to SVN Repository From Repositories View

I cannot connect to a SVN repository from the **Repositories** view of SVN Client. How can I find more details about the error?

First check that you entered the correct URL of the repository in the **Repositories** view. Also check that a SVN server is running on the server machine specified in the repository URL and is accepting connections from SVN clients. You can check that the SVN server accepts connections with the command line SVN client from CollabNet.

If you try to access the repository with a `svn+ssh` URL also check that a SSH server is running on port 22 on the server machine specified in the URL.

If the above conditions are verified and you cannot connect to the SVN repository please generate a logging file on your computer and send the logging file to support@oxygenxml.com. For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2
log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [%t] %c - %m%n
```

Restart the application, reproduce the error, close the application and send the file `logging.log` generated in the install directory to support@oxygenxml.com.

## Problem Report Submitted on the Technical Support Form

What details should I add to my problem report that I enter on the Technical Support online form of the product website?

For problems like server connection error, unexpected delay while editing a document, a crash of the application, etc for which the usual details requested on the Technical Support online form are not enough you should generate a log file and attach it to the problem report. In case of a crash you should also attach the crash report file generated by your operating system. For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2
log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [%t] %c - %m%n
```

Restart the application, reproduce the error and close the application. The log file is called `logging.log` and is located in the install folder.

## Signature verification failed error on open or edit a resource from Documentum

When I try to open/edit a resource from Documentum, I receive the following error:

signature verification failed: certificate for All-MB.jar.checksum not signed by a certification authority.

The problem is that the certificates from the Java Runtime Environment 1.6.0\_22 or later no longer validate the signatures of the UCF jars.

Set the `-Drequire.signed.ucf.jars=false` parameter, as explained in the [Setting a Parameter in the Launcher Configuration File / Startup Script](#) topic.

```
-vmargs
-Xms40m
-Xmx256m
-Drequire.signed.ucf.jars=false
```

## Cannot Cancel a System Shutdown

If I try to shutdown my Win XP when there is at least one modified document open in Oxygen XML Editor, I am prompted to cancel the shutdown or force quit the application. If I choose **Cancel**, the system shuts down anyway. Why is that?

This problem was reported on Windows XP systems only. The known workaround is to start Oxygen XML Editor using the `oxygen.bat` script.

## Compatibility Issue Between Java and Certain Graphics Card Drivers

Under certain settings, a compatibility issue can appear between Java and some graphics card drivers, which results in the text from the editor (in **Author** or **Text** mode) being displayed garbled. In case you encounter this problem, update your graphics card driver. Another possible workaround is, [open the Preferences dialog box](#), go to **Fonts > Text antialiasing**, and set the value of **Text antialiasing** option to ON.



**Note:** If this workaround does not resolve the problem, set the **Text antialiasing** option to other values.

## An Image Appears Stretched Out in the PDF Output

Sometimes, when publishing XML content (DITA, DocBook, etc), images are scaled up in the PDF outputs but are displayed perfectly in the HTML (or WebHelp) output.

PDF output from XML content is obtained by first obtaining a intermediary XML format called XSL-FO and then applying an XSL-FO processor to it to obtain the PDF. This stretching problem is caused by the fact that all XSL-FO processors take into account the DPI (dots-per-inch) resolution when computing the size of the rendered image.

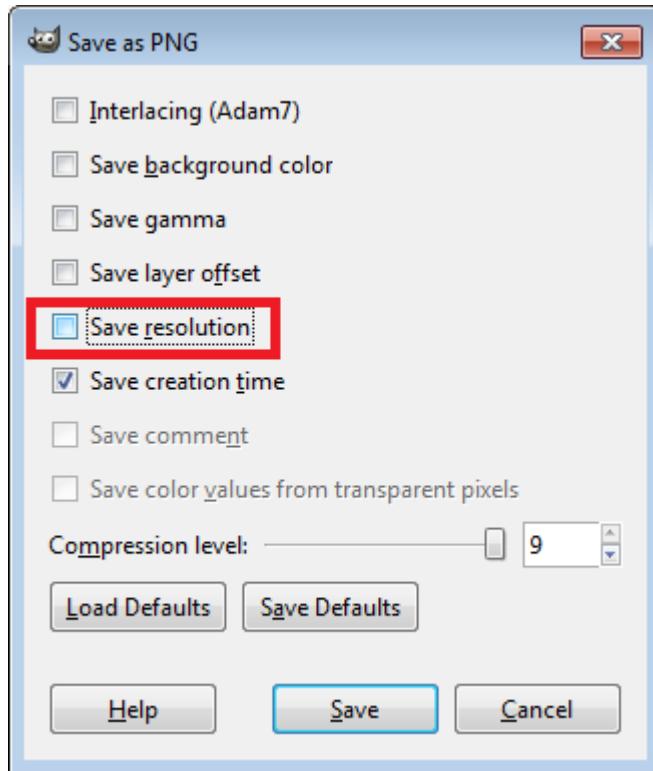
The PDF processor which comes out of the box with the application is the open-source Apache FOP processor. Here is what Apache FOP does when deciding the image size:

1. If the XSL-FO output contains width, height or a scale specified for the image `external-graphic` tag, then these dimensions are used. This means that if in the XML (DITA, DocBook, etc) you set explicit dimensions to the image they will be used as such in the PDF output.
2. If there are no sizes (width, height or scale) specified on the image XML element, the processor looks at the image resolution information available in the image content. If the image has such a resolution saved in it, the resolution will be used and combined with the image width and height in order to obtain the rendered image dimensions.
3. If the image does not contain resolution information inside, Apache FOP will look at the FOP configuration file for a default resolution. The FOP configuration file for XSLT transformations which output PDF is located in the `[OXYGEN_DIR]/lib/fop.xconf`. DITA publishing uses the DITA Open Toolkit which has the Apache FOP configuration file located in `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf`. The configuration file contains two XML elements called `source-resolution` and `target-resolution`. The values set to those elements can be increased, usually a DPI value of 110 or 120 should render the image in PDF just like in the HTML output.

The commercial **RenderX XEP** XSL-FO processor behaves similarly but as a fallback it uses 120 as the DPI value instead of using a configuration file.

**Tip:**

As a conclusion, it is best to save your images without any DPI resolution information in them. For example the open-source GIMP image editor allows you when saving a PNG image whether to save the resolution to it or not:



Having images without any resolution information saved in them allows you to control the image resolution from the configuration file for all referenced images.

## The DITA PDF Transformation Fails

To generate the PDF output, Oxygen XML Editor uses the DITA Open Toolkit.

If your transformation fails you can detect some of the problems that caused the errors by running [the Validate and Check for Completeness action](#). Depending on the options you select when you run it, this action reports errors such as topics referenced in other topics but not in the DITA Map, broken links, and missing external resources.

You can analyse the **Results** tab of the DITA transformation and search for messages that contain text similar to [ fop ] [ ERROR ]. If you encounter this type of error message, edit the transformation scenario you are using and set the **clean.temp** parameter to **no** and the **retain.topic.fo** parameter to **yes**. Run the transformation, go to the temporary directory of the transformation, open the **topic.fo** file and go to the line indicated by the error. Depending on the XSL FO context try to find the DITA topic that contains the text which generates the error.

If none of the above methods helps you, go to **Help > About > Components > Frameworks** and check what version of the DITA Open Toolkit you are using. Copy the whole output from the DITA OT console output and either report the problem on the DITA User List or to support@oxygenxml.com.

## The DITA to CHM Transformation Fails

Oxygen XML Editor uses the DITA Open Toolkit and the HTML Help compiler (part of the Microsoft HTML Help Workshop) to transform DITA content into *Compiled HTML Help* (or *CHM* in short).

It is a good practice to validate the DITA map before executing the transformation scenario. To do so, run [the Validate and Check for Completeness action](#). Depending on the selected options, this action reports errors, such as topics referenced in other topics (but not in the DITA Map), broken links, and missing external resources.

However, the execution of the transformation scenario may still fail. Reported errors include:

- [exec] HHC5010: Error: Cannot open "fileName.chm". Compilation stopped. - this error occurs when the CHM output file is opened and the transformation scenario cannot rewrite its content. To solve this issue, close the CHM help file and execute the transformation scenario again.
- [exec] HHC5003: Error: Compilation failed while compiling fileName - possible causes of this error are:
  - the processed file does not exist. Fix the file reference before executing the transformation scenario again.
  - the processed file has a name that contains space characters. To solve the issue, remove any spacing from the file name and execute the transformation scenario again.

## DITA Map ANT Transformation Because it Cannot Connect to External Location

The transformation is run as an external ANT process so you can continue using the application as the transformation unfolds. All output from the process appears in the **DITA Transformation** tab.

The HTTP proxy settings are used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the [the Proxy preferences page](#)

## Topic References outside the main DITA Map folder

Referencing to a DITA topic, map or to a binary resource (for example: image) which is located outside of the folder where the main DITA Map is located usually leads to problems when publishing the content using the DITA Open Toolkit. The DITA OT does not handle well links to topics which are outside the directory where the published DITA Map is found. By default it does not even copy the referenced topics to the output directory.

You have the following options:

1. Create another DITA Map which is located in a folder path above all referenced folders and reference from it the original DITA Map. Then transform this DITA Map instead.
2. Edit the transformation scenario and in the **Parameters** tab edit the **fix.external.refs.com.oxygenxml** parameter. This parameter is used to specify whether the application tries to fix up such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references. The fix up has no impact on your edited DITA content. Only "false" and "true" are valid values. The default value is false.

## The PDF Processing Fails to Use the DITA OT and Apache FOP

There are cases when publishing DITA content fails when creating a PDF file. This topic lists some common problems and solutions.

- The FO processor cannot save the PDF at the specified target. The console output contains messages like:

```
[fop] [ERROR] Anttask - Error rendering fo file: C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo <Failed to open C:\samples\dita\out\pdf\test.pdf>
Failed to open samples\dita\out\pdf\test.pdf
.....
[fop] Caused by: java.io.FileNotFoundException: C:\Users\radu_coravu\Desktop\bev\out\pdf\test.pdf
(The process cannot access the file because it is being used by another process)
```

Such an error message usually means that the PDF file is already opened in a PDF reader application. The solution is to close the open PDF before running the transformation.

- One of the DITA tables contains more cells in a table row than the defined number of *colspec* elements. The console output contains messages like:

```
[fop] [ERROR] Anttask - Error rendering fo file:
D:\projects\xml\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo
<net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException:
The column-number or number of cells in the row overflows the number of fo:table-columns specified for the
```

```
table. (See position 179:-1)>net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException: The
column-number or number of cells in the row overflows the number of fo:table-columns specified for the table.
(See position 179:-1)
[fop] at org.apache.fop.tools.attasks.FOPTaskStarter.renderInputHandler(Fop.java:657)
[fop] at net.sf.saxon.event.ContentHandlerProxy.startContent(ContentHandlerProxy.java:375)
.....
[fop] D:\projects\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo ->
D:\projects\samples\dita\flowers\out\pdf\flowers.pdf
```

To resolve this issue, correct the *colspec* attribute on the table that caused the issue. To locate the table that caused the issue:

1. Edit the transformation scenario and set the parameter *clean.temp* to *no*.
  2. Run the transformation, open the *topic.fo* file in Oxygen XML Editor, and look in it at the line specified in the error message (See position 179:-1).
  3. Look around that line in the XSL-FO file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.
- There is a broken link in the generated XSL-FO file. The PDF is generated but contains a link that is not working. The console output contains messages like:

```
[fop] 1248 WARN [main] org.apache.fop.apps.FOUserAgent - Page 6: Unresolved ID reference
"unique_4_Connect_42_wrongID" found.
```

To resolve this issue:

1. Use the  **Validate and Check for Completeness** action available in the **DITA Maps Manager** view to find such problems.
2. If you publish to PDF using a DITAVAL filter, select the same DITAVAL file in the **DITA Map Completeness Check** dialog box.
3. If the  **Validate and Check for Completeness** action does not discover any issues, edit the transformation scenario and set the *clean.temp* parameter to *no*.
4. Run the transformation, open the *topic.fo* file in Oxygen XML Editor, and search in it for the *unique\_4\_Connect\_42\_wrongID* id.
5. Look around that line in the XSL-FO file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

## The *TocJS* Transformation Doesn't Generate All Files for a Tree-Like TOC

The *TocJS* transformation of a DITA map does not generate all the files needed to display the tree-like table of contents. To get a complete working set of output files you should follow these steps:

1. Run the *XHTML* transformation on the same DITA map. Make sure the output gets generated in the same output folder as for the *TocJS* transformation.
2. Copy the content of  
[OXYGEN\_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/basefiles folder in the transformation's output folder.
3. Copy the  
[OXYGEN\_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/sample/basefiles/frameset.html file in the transformation's output folder.
4. Edit frameset.html file.
5. Locate element <frame name="contentwin" src="concepts/about.html">.
6. Replace "concepts/about.html" with "index.html".

## Navigation to the web page was canceled when viewing CHM on a Network Drive

When viewing a CHM on a network drive, if you only see the TOC and an empty page displaying “Navigation to the web page was canceled” note that this is normal behaviour. The Microsoft viewer for CHM does not display the topics for a CHM opened on a network drive.

As a workaround, copy the CHM file on your local system and view it there.

## Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x

On some Linux systems based on Gnome 3.x (e.g. Ubuntu 11.x, 12.x) the main menu of Oxygen XML Editor has alignment issues when you navigate it using your mouse.

This is a known problem caused by Java SE 6 1.6.0\_32 and earlier. You can resolve this problem using the latest Java SE 6 JRE from Oracle. To download the latest version, go to

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

To bypass the JRE bundled with Oxygen XML Editor, go to the installation directory of Oxygen XML Editor and rename or move the `jre` folder. If Oxygen XML Editor does not seem to locate the system JRE, either set the `JAVA_HOME` environment variable to point to the location where you have installed the JRE, or you can simply copy that folder with the JRE to the installation directory and rename it to `jre` to take the place of the bundled JRE.

## JPEG CMYK Color Space Issues

JPEG images with CMYK color profile having the color profiles embedded in the image should be properly rendered in the **Author** mode.

If the color profile information is missing from the JPEG image but you have the ICC file available, you can copy the `profileFileName.icc` to the `[OXYGEN_DIR]\lib` directory.

If the color space profile is missing, JPEG images that have the CMYK color space are rendered without taking the color profile into account. The **Unsupported Image Type** message is displayed above the image.

## SVG Rendering Issues

Oxygen XML Editor uses the **Apache Batik** open source library to render SVG images. The **Batik** library only has partial support for SVG 1.2: <http://xmlgraphics.apache.org/batik/dev/svg12.html>.

For example, if you are using the *Inkscape* SVG editor, it is possible that it saves the SVG as 1.1 but it actually uses SVG 1.2 elements like `flowRoot` inside it. This means that the image will not be properly rendered inside the application.

SVG images shown in the **Author** visual editing mode are rendered as static images, without support for animations and Javascript.

## MSXML 4.0 Transformation Issues

In case the latest MSXML 4.0 service pack is not installed on your computer, you are likely to encounter the following error message in the **Results** panel when you run a transformation scenario that uses the MSXML 4.0 transformer.

### Error Message

```
Could not create the 'MSXML2.DOMDocument.4.0' object.
Make sure that MSXML version 4.0 is correctly installed on the machine.
```

To fix this issue, go to the Microsoft website and get the latest MSXML 4.0 service pack.

---

# Chapter

# 23

---

## Using the Oxygen XML WebApp

---

**Topics:**

- [Oxygen XML WebApp Overview](#)
- [License Issues](#)

Oxygen XML WebApp web editing platform leverages the advanced oXygen XML authoring technology to bring XML editing and reviewing to your mobile devices, as well as your desktop systems. The innovative mobile-first design of the user interface allows you to interact with your XML content like never before.

Oxygen XML WebApp is an independent product, designed to complement the oXygen XML suite. You can give it a try by [installing it as an add-on](#). However, to be able to use it in a production environment, it needs [server deployment](#) and a dedicated Oxygen XML WebApp license.

## Oxygen XML WebApp Overview

The interface groups the available actions in the following functional areas:

- On mobile devices the upper toolbar allows you to switch between the **Edit** and **Review** modes.
- The actions on the upper toolbar include **Undo**, **Redo**, **Save**, **Find/Replace**, and **Validation Results**.
- The lower toolbar provides document-specific actions. For example, for DITA documents, the **DITA** toolbar presents DITA-specific actions. Also, it includes the **Review** toolbar that presents actions such as **Track Changes**, **Add Comment**, and more.
- The side panel includes tabs to switch between the **Review**, **Attributes**, and **Find/Replace** views.

 **Note:** Some documents might be locked for editing. This means that you cannot alter their content or add comments to them. A document lock state is indicated by a padlock icon next to the file name.

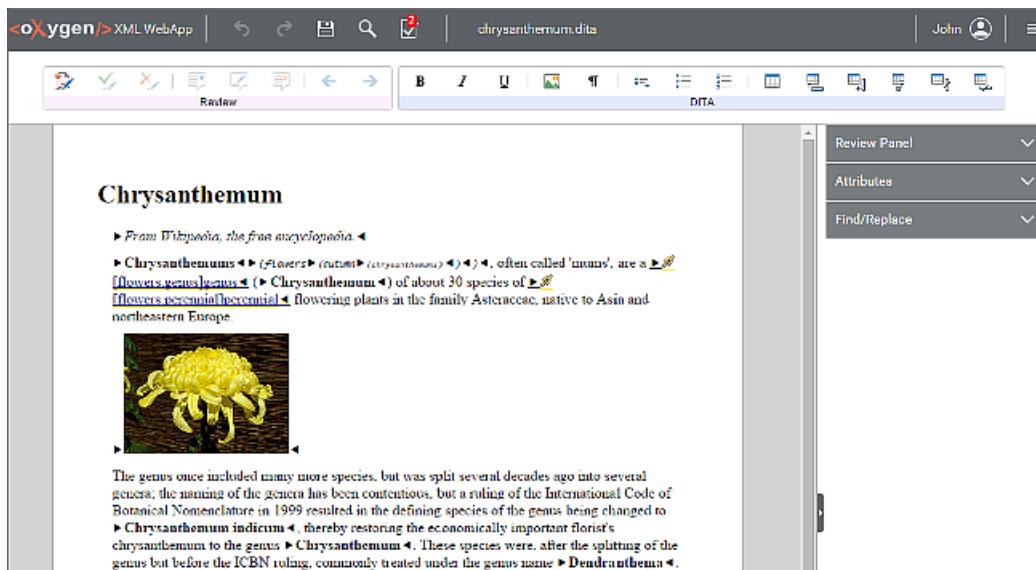
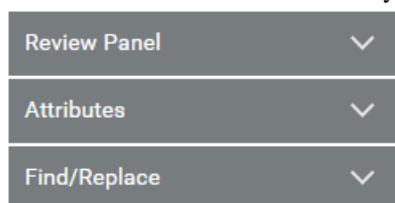


Figure 443: The Desktop Version of the Oxygen XML WebApp

### Editing Actions

The WebApp Reviewer allows you to not only review documents, but to also make changes to them. To activate this mode on mobile devices, tap or click the **Edit** button in the upper toolbar. Also, in mobile devices if you swipe left, or use the  **Menu** button, a side panel is displayed at the right side of the editing area. This panel provides you with access to several views and a variety of editing actions.



The following views can be accessed by using the options in the side panel on the right side of the editor:

- Review Panel** - Displays the tracked changes and comments made to the document by the content authors.
- Attributes** - Displays all possible attributes and their value. You can also edit attribute values.
- Find/Replace** - Provides **Find** and **Replace** actions for searching the current document.

In a desktop web browser, the contextual menu also offers the following actions, along with framework-specific actions:

#### Undo

This action is available in the contextual menu only after you make a change.

**Redo**

This action is available in the contextual menu only after you use the **Undo** action.

**Insert Element**

Inserts an element at current position.

**Rename Element**

Renames the current element.

**Add Comment**

Allows you to insert a comment on a selected fragment of text.

**Framework Specific Actions**

Depending on the type of document you are editing, the toolbar contains a series of actions defined at framework level. Oxygen XML WebApp comes with built-in actions for DITA, DocBook, TEI, and XHTML document types. These actions include:

**Styling actions**

Most common styling actions, such as **Bold**, **Italic**, **Underline**.

**Insert actions**

Actions that allows you to insert a variety of framework-specific objects, such as images, tables, paragraphs, and other elements.

**List actions**

Actions that allow you to create ordered and unordered lists, and add new items in them.

**Table actions**

Actions that allow you to create a table and manage its structure (insert and delete rows and columns).

**How to Perform Common Edit Tasks****Input text:**

1. Tap or click to move the caret into the selected document area.
2. Type the text using the keyboard

**Split a paragraph:**

1. Tap or click to move the caret at the split position.
2. Press Enter on the keyboard to display the *Content Completion Assistant*. Choose *Split p* from the list of proposals.

**Enter a new XML element, or to surround the selection in an XML element**

1. Tap or click to move the caret into the selected document area, or make a text selection.
2. Tap or press **Enter** to open the **Content Completion Assistant**.
3. Select the element name.

**Activate change tracking**

1. Tap or click the  **Track Changes** button from the **Review** toolbar to enable or disable this feature.

**Set an attribute value**

1. Tap or click the  **Menu** button from the upper right side corner, or swipe from right to left. A side panel is displayed.
2. Press the **Attributes** tab. The attributes side panel is displayed.

Attributes	
Attribute	Value
class	+ topic/ph hi-d/b
See all attributes...	

- If the attribute is present in the attributes table (it already has a value), you can change its value by tapping/clicking its value. If the attribute is not shown, press the **See all attributes** button. Tap or click the attribute to set its value

### Remove an attribute

- Tap or click the  **Menu** button from the upper right side corner, or swipe from right to left. A side panel is displayed.
- Press the **Attributes** tab. The attributes side panel is displayed.
- Find the attribute in the table and press or tap its value.
- Press **Remove Attribute** in the subsequent dialog box.

### Save a modified XML document

To save your changes, tap or click the  **Save** button from the upper toolbar. Note that the open and save operations depend on the integration of the Oxygen XML WebApp with a CMS or other storage mechanism.

### Search and replace content

- Tap or click the  **Find/Replace** button from the upper toolbar, or the **Find/Replace** tab.
- The **Find/Replace** side panel is open. Type the text you want to find in the **Search for** input box and press the  **Find** icon in the left side of that same input box.
- All matches are highlighted, and the first one is selected. To advance to the next match, press the  **Find** icon again.
- To replace content, type the new text in the **Replace with** input box. To replace the current match, press the **Replace** button. To replace all matches, press the **Replace all** button.

### Checking an XML Document for Errors

Oxygen XML WebApp automatically checks the document for errors. The errors and warnings reported by the validation engine include problems found in the following:

- The structural integrity and well-formedness of the document.
- A set of best practice rules (such as a maximum number of words in a paragraph).

The automatic validation errors appear underlined in red, while warnings underlined in yellow. On the desktop version, if you hover the mouse over the errors or warnings, a tooltip is displayed that provides more information about the problem.

 To see a detailed list of errors, tap or click the **Validation Results** button on the upper toolbar (note the red marker that shows the number of errors found in the document). Every item in the list has an arrow button to its right side. Tap or click that button to go to the location in the document where the issue was found.

### Copy-Paste Support

The Oxygen XML WebApp includes support for copy and paste actions, including:

- From external sources (such as text processors or web browsers) to the document you are editing. Oxygen XML WebApp also tries to *preserve all associated formatting*, such as lists, paragraphs, and text styling.
- Within a WebApp session, preserving the XML structure.
- From the WebApp session to external sources. In this case, only the text content is copied.



**Note:** On Safari Mobile versions 6 and 7, the copy/paste support ignores all text formatting, keeping only the content.

The *copy-paste* support does not have dedicated actions in the toolbar or context menu due to security restrictions imposed by most web browsers.

- On desktop browsers, the *copy-paste* support is available through the usual keyboard shortcuts **Ctrl C (Command C on OS X)** for copying, **Ctrl X (Command X on OS X)** for cutting, and **Ctrl V (Command V on OS X)** for pasting).
- On mobile browsers, you can use the usual actions specific to each platform.

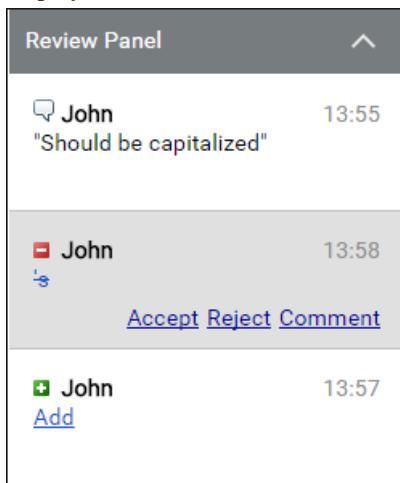
## How to Perform Common Review Tasks

### Add a comment:

1. Make a selection in the document content.
  2. Tap or click the **Add comment** button from **Review** toolbar.
  3. Input a comment in the displayed dialog box. Press **Comment** to commit it into the document.
- Note:** To modify one of your comments, select the comment and press the **Change** button.
4. The commented text is highlighted.

### See the list of all comments or tracked changes from your document:

1. Tap or click the **Review Panel** tab from the upper right side corner, or swipe from right to left. A side panel is displayed with all the comments and tracked changes.



2. Tap or click one of the comments or tracked changes to see it highlighted in the document area.
3. Hover over a comment to display the **Remove** and **Change** options. For the tracked changes, the list of available actions are: **Accept**, **Reject**, and **Comment**.

### Navigate through the comments or tracked changes:

1. Tap or click the left and right arrow buttons ( ) from the **Review** toolbar.
2. The current comment or tracked change is highlighted in the document. Also, it is displayed in the right-side **Review panel**.

## Browser Compatibility

Oxygen XML WebApp was developed and continuously tested on the following major Web browsers:

- Internet Explorer 9 and newer, running on desktop systems.

- Opera 15 and newer, running on desktop systems.
- Chrome 20 and newer, running on desktop systems.
- Mozilla Firefox 19 and newer, running on desktop systems.
- OS X Safari 6, running on OS X.
- Safari Mobile iOS 6, running on iOS devices.
- Chrome for Android 4.3 and newer, running on Android-enabled devices.

As an HTML 5 application, it is likely to work on other HTML 5 compliant browsers for various platforms.

## Known Issues

Due to implementation particularities, Oxygen XML WebApp may exhibit minor behavioural differences:

- On Android devices the content completion list of proposals might display *undefined* elements. To prevent this, go to **Settings > Bandwidth Management > Reduce data usage** and select **OFF**.
  - On Safari Mobile and Chrome for Android, there is no warning message if you close the browser page without saving the changes made in the document.
  - *Input Method Editor (IME)* is fully supported only when running Oxygen XML WebApp in a Chrome browser on a Windows platform.
  - On Safari Mobile, the native **Bold**, **Italic**, and **Underline** actions do not work. As a workaround, use the framework-specific markup.
  - On Android devices, the editing works best with *Google Keyboard* having the **Auto-correction** option disabled and the **Show correction suggestions** option set to **Always hidden**. Alternatively, you can use *Google voice typing*.
-  **Note:** Using other keyboards can lead to unpredictable results. If your document gets corrupted, use the **Undo** button.
- On Safari Mobile versions 6 and 7, the copy/paste support ignores all text formatting, keeping only the content.

## License Issues

---

Oxygen XML WebApp uses a floating license mechanism, where the license key is stored on a server and individual users consume license seats from a common pool. To run properly, be aware of the following:

- In order for the licensing mechanism to run properly, your browser needs to accept cookies. Otherwise, Oxygen XML WebApp will not be loaded.
- Each browser consumes a license. When you use multiple different browsers (for example, Firefox and Chrome) to access the Oxygen XML WebApp at the same time on the same system, you will consume multiple licenses. However, multiple tabs in the same browser consume a single license.
- A license is automatically released after 30 minutes of inactivity. When resuming work, if there are no available licenses, you can still save the documents you are currently working on.
- On desktop systems, you may force an immediate license release by closing all editor tabs.

---

# Chapter

# 24

---

## Customizing Oxygen XML WebApp

---

**Topics:**

- [\*Customization Overview\*](#)
- [\*Deploying Oxygen XML WebApp\*](#)
- [\*Licensing the Oxygen XML WebApp\*](#)
- [\*Oxygen XML WebApp How To\*](#)

This section describes the various ways that you can customize the Oxygen XML WebApp.

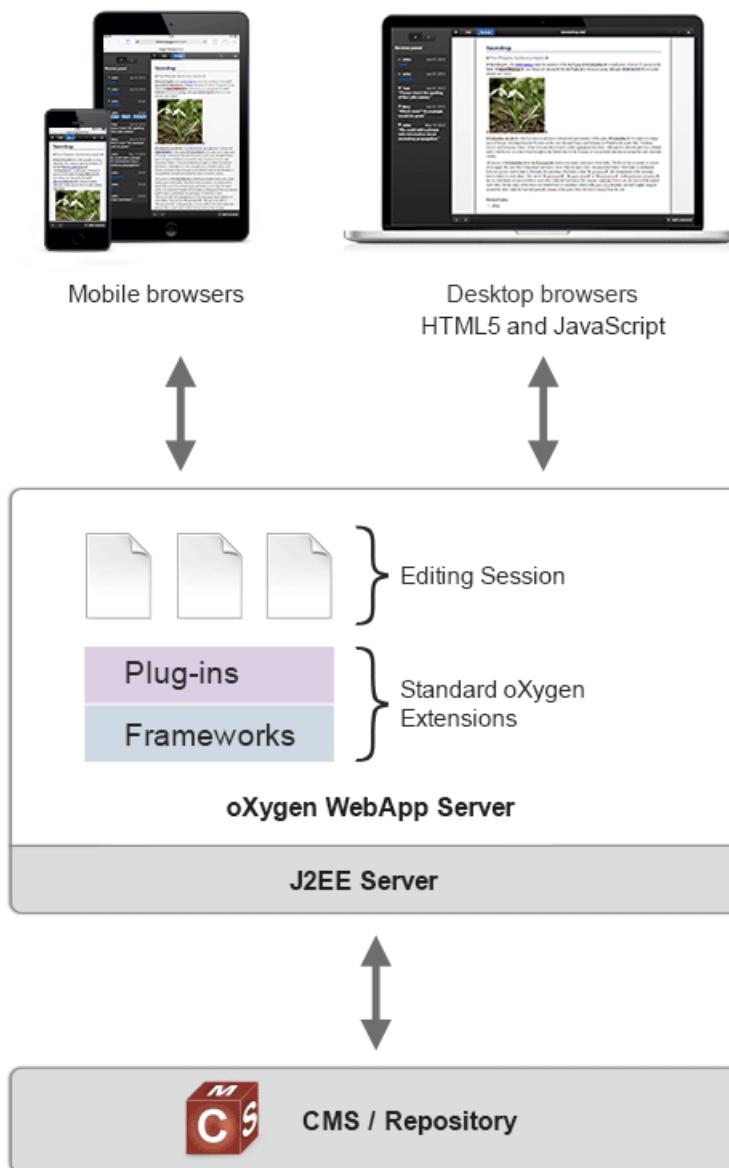
## Customization Overview

The core of oXygen XML Author can be deployed on a server, allowing a variety of HTML5-enabled client devices to edit and review XML content.



**Note:** Despite the efforts spent to ensure that the frameworks, options, and plugins behave similarly in both WebApp and standalone version of oXygen XML Author, there might be some differences imposed by specific platform limitations.

### A Graphical Description of the WebApp System



Oxygen XML WebApp was designed to accommodate a large degree of customization.

## Options

Functionality common with the standalone distribution of oXygen XML Author, as they share the same options. This allows you to configure a consistent editing experience for all users.

### Documentation Frameworks

Custom documentation frameworks can be re-used between the oXygen XML Author standalone distribution and the Oxygen XML WebApp.

-  **Note:** Oxygen XML WebApp comes bundled with specially tuned frameworks for DITA, DocBook, TEI, and XHTML document types. Any other framework from the standalone distribution can also be customized and used in Oxygen XML WebApp.

### Server Side

The Oxygen XML WebApp server side can be customized by using the following plugin types:

- The **URLStreamHandler plugins** can be used to integrate the WebApp with CMS or XML databases.
- The **WorkspaceAccess plugins** that provides access to the oXygen plugin-level Java API.

### Client Side

Client side customization is available through a JavaScript API. Unlike the server side customization, it can be used to modify the application's GUI.

## Customizing Oxygen XML WebApp Options

### Author Options

You can modify the options in the oXygen XML Author standalone application and then export them as an XML file by clicking the **Options > Export Global Options...** menu action. The exported options file should replace the `options.xml` file from `bundle-options/oxygen-options/` folder in the [oXygen XML SDK project](#).

### WebApp Configuration File

A small number of options are specific only to the Oxygen XML WebApp and they can be configured in the `WEB-INF/web.xml` file. Each option is specified as a `context-param` element.

The following is a list of options and their accepted values:

Option name	Value	Description
<code>com.oxygenxml.loadBuiltInProtocolHandlers</code>	true/false	Controls whether or not the oXygen built-in handlers for HTTP/HTTPS and FTP/SFTP protocols are installed. Default value is true.
<code>com.oxygenxml.webapp.datastore.docs.memory.size</code>	An integer number.	Indicates the number of editing sessions stored in memory.
<code>com.oxygenxml.webapp.datastore.docs.memory.expire</code>	Duration (*)	Indicates the delay after which inactive sessions are stored on disk.
<code>com.oxygenxml.webapp.datastore.docs.disk.size</code>	An integer number.	Indicates the number of inactive editing sessions that can be stored on disk.
<code>com.oxygenxml.webapp.datastore.docs.disk.expire</code>	Duration (*)	Indicates the delay after which inactive sessions are discarded.

(\*) - Duration is represented by an integer, followed by one of "d", "h", "m", or "s", representing days, hours, minutes, or seconds, respectively.

Here is an example of how to configure a context parameter:

```
<context-param>
 <param-name>com.oxygenxml.loadBuiltInProtocolHandlers</param-name>
 <param-value>false</param-value>
</context-param>
```

## Customizing Oxygen XML WebApp Documentation Frameworks

The custom documentation frameworks can be re-used between the oXygen XML Author standalone distribution and the WebApp, but some fine tuning might be necessary to achieve a better editing experience. The advantages of using a common framework include:

- Easier development and testing, since you can test most of the functionality in the standalone version of oXygen XML Author, using advanced tools such as the **CSS Inspector**, **CSS Editor**, or the **Document Type** customization dialog box.
- Uniform experience across different oXygen XML Author distributions.
- Reuse previously developed frameworks.

### Developing And Testing a Documentation Framework Using the WebApp Add-on

The following procedures assumes that you have access to an oXygen XML Author standalone installation. This is not a mandatory requirement, but a way to speed-up the development process.

1. Use the standalone installation of oXygen XML Author to customize the documentation framework. Modifications made to the framework are instantly visible, but if you want to preview them in the WebApp, proceed to the next step.
2. *Run the Oxygen XML WebApp using the add-on distribution and test the documentation framework.*



**Note:** The changes that you make to your documentation framework will not be automatically reflected in the running WebApp. To see the results of changes, close the server using the **Close and stop server** button and start it again.

### Deploying a Documentation Framework

1. Copy your customized framework into the `bundle-frameworks/oxygen-frameworks/` folder of the *oXygen XML SDK project*.
2. *Build the SDK project and deploy it.*

### Customization Tips

- If you want to use CSS rules that only apply when the framework is used in the Oxygen XML WebApp, use the following media query:

```
@media oxygen AND (platform:webapp) {
 ...
}
```

- In every framework, you can add a `web/framework.js` file that uses the *JavaScript API* to implement editing actions and add them to the toolbar and contextual menu.
- If the framework contains Oxygen XML Author operations (Java implementations of the `ro.sync.ecss.extensions.api.AuthorOperation` interface), they can be enabled to be used by the Oxygen XML WebApp by using the `ro.sync.ecss.extensions.api.WebappCompatible` annotation.



**Note:** Oxygen XML Author operations that use *Java Swing* components to display a graphical interface to the user are not compatible with Oxygen XML WebApp, so they should not be annotated.

- Oxygen XML WebApp continuously validates the XML documents using the default validation scenarios defined at framework level. Only the validation units that are marked for **Automatic Validation** will be used. You can use the **Document Type** customization dialog box to configure the automatic validation in the WebApp.

### Oxygen XML WebApp CSS Limitations

Oxygen XML WebApp CSS support is compatible with that offered by the standalone distribution of oXygen XML Author, with the following exceptions:

- The `+ (direct adjacent)` and `> (child selector)` structural selectors cannot be used to match table-related elements.
- Oxygen CSS extensions are ignored on `print` media. If an Oxygen CSS extension is used on the `screen` media, it will also be used on the `print` media.
- Oxygen CSS extension `properties` and `functions` cannot be used in a rule that has a `:hover` pseudo-class in the selector. The `attr` function is also not supported in such a rule due to a lack of browser support.
- The `:hover` pseudo-class is only available for mouse-enabled platforms.
- Oxygen CSS extensions used in property values that express lengths may not behave as expected. Nevertheless, it is a good approximation.
- Oxygen synthetic DOM nodes `comment`, `reference`, `cdata`, `pi`, and `error` interfere with the `+ (direct adjacent)` structural selector. For example:

```
b + b {
 color: red;
}
```

will not match the following XML structure:

```
<root>

 <!--comment-->

</root>
```

- Oxygen XML WebApp does not render non-table-row children elements of tables and non-table-cell elements of table-row elements.
- Oxygen XML WebApp does not support:
  - `:nth-last-of-type`, `:first-of-type`, `:last-of-type`, `:nth-last-of-type` pseudo-classes.
  - Subject selectors, since they are not supported by web browsers.
  - Specifying widths for inline elements.
  - Attribute selectors that use wildcard for the attribute name.
  - Oxygen CSS extensions to style `:before` and `:after` pseudo-elements, except in the `content` property.
  - CSS property values that contain the `oxy_xpath` function; they are not refreshed correctly.
  - Registering a `ro.sync.ecss.extensions.api.StylesFilter`; it is ignored.

### Oxygen XML WebApp Editor Variables

Oxygen XML WebApp processes oXygen editor variables. However, the following categories of editor variables are not supported:

- Editor variables related with functionality that is not available in the Oxygen XML WebApp, such as  `${dbgXML}`  or  `${dbgXSL}` .
- Editor variables related with oXygen project location, such as  `${pdu}` ,  `${pd}` , or  `${pn}` .
- Any editor variable that displays Java Swing-based components, such as  `${ask}` .
- Editor variables related with the oXygen standalone installation directory, such as  `${oxygenHome}`  or  `${oxygenInstallDir}` .

## Customizing Oxygen XML WebApp Plugins

We currently provide support for the following extension types:

1. The **URLStreamHandler extensions** can be used to integrate the WebApp with CMS-es or XML databases. There is an example URLStreamHandler provided in [oxygen XML SDK project](#) in the **oxygen-sample-plugins/oxygen-sample-plugin-custom-protocol** folder. The extension uses the **cproto** protocol to access the file system of the server and can be used as a starting point.
 

 **Note:** For more details about implementing an authentication mechanism, see the [How To Make WebApp Use the CMS Authentication Mechanism](#) on page 1122 topic.
2. In the **WorkspaceAccess extensions** most of the methods used to configure the oXygen GUI are unavailable, but the extensions can still be used, for example, to configure a `javax.xml.transform.URIResolver`.
 

 **Note:** The `ro.sync.ecml.workspace.api.PluginWorkspace` instance passed to the extension also implements the `ro.sync.ecss.extensions.api.webapp.access.WebappPluginWorkspace` interface and provides access to some Oxygen XML WebApp-specific functionality.
3. The **WebappServlet** extension allows you to provide an implementation of a servlet-like interface (`ro.sync.ecss.extensions.api.webapp.plugin.WebappServletPluginExtension`) that will be dynamically loaded by the WebApp. Your implementation will also provide the path to the location where the servlet will be exposed.

### Loading plugin-related custom JavaScript code

If your plugin needs accompanying JavaScript code to be loaded and executed on the client-side you can bundle it together with your plugin code. Oxygen XML WebApp loads all files with the `.js` extension located in the `web` folder of the plugin.

### Adding the plugins in the WebApp

If you have already developed such oXygen plugins, they can be added in the **bundle-plugin/dropins** folder in the Maven project.

If you are developing a new oXygen plugin you are encouraged to use as a starting point any of the existing plugins. Then you should add the resulting Maven project as a dependency (or even a sub-module) in the **oxygen-sample-plugins** module.

## Customizing Oxygen XML WebApp's Client Side

Oxygen XML WebApp is an editing platform, but it is the integrator's job to provide a way for the user to select/choose which file is going to be edited. Afterwards, the user should be redirected to WebApp's editing page, along with three URL parameters:

- `url` - absolute URL of the edited file
- `ditamap` - absolute URL, optional parameter. Taken into account only when editing a DITA file. Provides the DITA map context of the edited DITA file.
- `author` - author name

### Example

Let's suppose that the WebApp is deployed at the following URL:

```
http://www.example.com/oxygen-sdk-sample-webapp/
```

The user (whose name is John Doe) wants to edit a file (located at `http://www.test.com/topics/topic.xml`) in the context of a DITA map (located at `http://www.test.com/map.xml`). In this case, the editing URL should be:

```
http://www.example.com/oxygen-sdk-sample-webapp/app/demo-mobile.html?
url=http%3A%2F%2Fwww.test.com%2Ftopics%2Ftopic.xml&
ditamap=http%3A%2F%2Fwww.test.com%2Fmap.xml
&author=John%20Doe
```



**Note:** The parameters values are percent encoded before being added to the editing URL.

## Loading Custom JavaScript Code

To extend the functionality provided by Oxygen XML WebApp, create a file called `plugin.js` and copy it in the `app` folder of the WebApp deployment. *Alternatively, you can bundle JavaScript code with a Java Plugin.*

The `plugin.js` file can contain JavaScript code that calls the *JavaScript API* provided by Oxygen XML WebApp.

# Deploying Oxygen XML WebApp

---

## Server Requirements

Even though there are not very strict requirements, you should consider the following metrics when provisioning the server for running the Oxygen XML WebApp:

- a processor core can handle 50 to 100 active users.
- editing an average DITA file consumes about 10MB of RAM. However, Oxygen XML WebApp provides a *configurable mechanism* that, under memory pressure, stores on disk the least recently used files.

## Software Requirements

On the server side, the following applications are supported:

- Apache Tomcat 7 or 8.
- Java Virtual Machine 1.7 or newer.

# Licensing the Oxygen XML WebApp

---

Oxygen WebApp uses a floating license model, where the license key is stored on a server and individual users consume license seats from a common pool.

## How it works

The license key contains the maximum number of users that can simultaneously access the WebApp at any given moment. After a period of inactivity, the license allocated to that user becomes available.

While no personal information is sent to the server, a cookie that identifies the user is auto-generated. Note that the use of two different browsers (for example, Firefox and Chrome) by a single user, will consume two floating licenses. However, using two or more windows or tabs of the same browser, consumes a single floating license.

## Licensing

Follow these steps to license a deployment of Oxygen XML WebApp:

1. To obtain a license key, please contact [support@oxygenxml.com](mailto:support@oxygenxml.com).
2. *Install a floating license server.* If you decide to use an HTTP license server, you can deploy it in the same Tomcat server, alongside with Oxygen XML WebApp.
3. Configure the license server connection.

## Configuring the license server

The connection to the server should be configured in a file located at `WEB-INF/license.properties`. It should have the following keys.

### `licensing.server.type`

Type of licensing server. Can be one of `http` or `standalone`.

For an HTTP server, configure the following parameters:

### `licensing.server.url`

The URL of the license server

### `licensing.server.user`

The user name used for the license server

### `licensing.server.password`

The password used for the license server

For a standalone server, configure the following parameters:

### `licensing.server.host`

The host name of the licensing standalone server

### `licensing.server.port`

The port of the licensing standalone server

A configuration file might look like this:

```
licensing.server.type=http
licensing.server.url=http://example.com:8080/oxygenLicenseServlet/license-servlet
licensing.server.user=admin
licensing.server.password=*****
```

---

## Oxygen XML WebApp How To

This section covers a variety of common use cases.

### How To Share a Tomcat Instance Between Oxygen XML WebApp And Another Application

Due to a class loader issue, the oXygen XML built-in protocol handlers cannot be used in a scenario where the WebApp shares the same Apache Tomcat instance with another application. To disable the protocol handlers initialization, set the `com.oxygenxml.loadBuiltInProtocolHandlers` option to `false`.

Also, the following issues need to be considered:

- oXygen reads and sets system properties, and while we try to namespace oXygen-specific ones, there is no guarantee that there won't be any clashes with those set by other applications.
- you have to adapt the JVM's memory configuration to the scenario where there will be more applications competing for the same pool of memory.
- oXygen XML Author WebApp currently does not restart (or *reload* in Apache Tomcat terminology) correctly unless the Apache Tomcat server is also restarted.

### How To Make WebApp Use the CMS Authentication Mechanism

This topic covers the case when you want to impose an authentication step to all users who want to edit documents in Oxygen XML WebApp. This is usually required when the CMS needs authentication before granting access to a file.

Oxygen XML WebApp provides both server-side and client-side API that allows you to implement such a mechanism. The following is a list of the basic building blocks of the authentication mechanism:

1. Develop a plugin that implements the

`ro.sync.xml.plugin.urlstreamhandler.URLStreamHandlerPluginExtension` interface.

Considering the multi-user context of the WebApp, the `getURLStreamHandler` method should return an instance

of the `ro.sync.ecss.extensions.api.webapp.plugin.URLStreamHandlerWithContext` class.

This class tracks the user on behalf of which the URL connection will be made.

2. If the CMS rejects the connection attempt with a message that the user is not authenticated, you should throw a `ro.sync.ecss.extensions.api.webapp.plugin.UserActionRequiredException` exception. This exception is automatically relayed to the client-side as a `sync.api.WebappMessage` JavaScript object.
3. On the client side:
  - Use the `sync.api.Editor.EventTypes.CUSTOM_MESSAGE_RECEIVED` event to intercept the messages sent from the server-side.
  - Display a dialog box to collect more authentication information from the user.
  - Send the credentials to the server, more specifically to the `ro.sync.ecss.extensions.api.webapp.plugin.URLStreamHandlerWithContext` instance defined at step 1. For this part, you will need to implement a secure way to transmit the credentials. This can range from a simple servlet that runs in the WebApp to an *OAuth* implementation.
  - Retry the operation that triggered the authentication procedure.

## How To Configure WebApp Minimal File Access Permissions

WebApp requires access to the following file resources:

- *READ* access to the directory where the WebApp is deployed.
- *READ* and *WRITE* access to the application's working directory.
- *READ* and *WRITE* access to JVM's temporary directory.

It is a good security practice to allow a component to access only the information and resources that are necessary for its purpose. In an environment that uses Apache Tomcat, you can enforce these rules following these steps:

- Start the Apache Tomcat server using the `-security` flag.
- Edit the `catalina.policy` file and add the following snippet:

```
grant codeBase "file:${catalina.base}/webapps/oxygen-webapp/-" {
 // Oxygen uses System properties for various configuration purposes.
 permission java.util.PropertyPermission "*", "read,write";
 // Oxygen custom protocols need access to network.
 permission java.net.NetPermission "*";
 permission java.net.SocketPermission "*", "accept,connect,listen,resolve";
 // The web framework used by Oxygen Webapp uses reflection and classloaders.
 permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
 permission java.lang.RuntimePermission "*";

 // Oxygen uses the JVM's java.io.tmpdir for various file handling tasks.
 permission java.io.FilePermission "${java.io.tmpdir}/-", "read,write,delete";
};
```

 **Note:** In the previous example, in the first line, replace `oxygen-webapp` with the name of your deployment of Oxygen XML WebApp.

## Configuring File Permissions to Custom Locations

There are cases when Oxygen XML WebApp needs to access files system resources, but, due to security reasons, you want to prevent your users to open them directly in the WebApp's editing page using the `file://` protocol.

You can do this by following these steps:

- Edit the `catalina.policy` file and add a line such as:

```
permission java.io.FilePermission "path/to/yourSecretDir/-", "read,write,delete";
```

- Use the following system property when starting the Tomcat server:

```
-Dfile.protocol.blacklist=/path/to/yourSecretDir
```



**Note:** Use the value of `path.separator` system property to separate more directories. For example, under Linux, the value of `path.separator` property is a colon punctuation character `:`.

## How To Use the WebApp With an WebDAV Server

The [oXygen XML SDK project](#) includes a WebDAV plugin that enables you to access files stored on a WebDAV server. Follow these steps:

1. Create a sample project following the procedure available on [oXygen XML SDK project](#) website.
2. In order to license the Oxygen XML WebApp component, follow the instructions given [here](#).
3. [\*Run the WebApp instance\*](#).
4. You can now open a file stored on a WebDAV server. To open an WebApp session, you need to pass the file's URL prefixed with `webdav-` as the value of [\*the url parameter\*](#), like  
`webdav-https://exampleServer.com/file.xml`

---

# Chapter

# 25

---

## Comparison Between oXygen XML Author Component and Oxygen XML WebApp

---

The Author Component was designed to provide the functionality of the standard **Author** mode, which can be embedded either in a third-party standalone Java application or customized as a Java Web Applet to provide WYSIWYG-like XML editing directly in your choice of web browsers.

Oxygen XML WebApp is a re-implementation of the oXygen **Author** mode user interface, based on JavaScript and HTML5. Its purpose is to enable XML editing and reviewing on your mobile devices and desktops, directly in a web browser environment. Since the interface was thinned down as much as possible, the core XML processing was moved into a Java-enabled server.

Considering the particularities of these two approaches, a number of differences can be observed:

	Oxygen XML WebApp	oXygen XML Author Component Applet
Intended audience	Reviewers and occasional contributors.	Authors, technical writers.
Mobile device support	Specifically designed for mobile devices.	No.
Compatibility with the standard version of oXygen XML Author	Covers only editing and reviewing features.	100%
Text and Grid Mode	No.	Yes.
Client-side setup	None.	Requires Java to be installed, and Java Applets to be allowed to run.
Server-side setup	Requires a servlet container.	Requires a web server.



# Glossary

---

## Previous Topic

[Comparison Between oXygen XML Author Component and Oxygen XML WebApp](#)

## Java Archive

---

JAR (Java ARchive) is an archive file format. JAR files are built on the ZIP file format and have the .jar file extension. Computer users can create or extract JAR files using the `jar` command that comes with a JDK.

Java Archive (JAR)

### JAR

## Apache Ant

---

Apache Ant (Another Neat Tool) is a software tool for automating software build processes.

### Ant

## Active cell

---

The selected cell in which data is entered when you begin typing. Only one cell is active at a time. The active cell is bounded by a heavy border.

## Block element

---

A block element is an one that is intended to be visually separated from its siblings, usually vertically. For instance, a paragraph or a list item are block elements. It is distinct from a *inline element* which has no such separation.

## Inline element

---

An inline element is one that is intended to be displayed in the same line of text as its siblings or the surrounding text. For instance, strong and emphasis in HTML are inline elements. It is distinct from a *block element*, which is visually separated from its siblings.

## DITA map

---

A DITA map is a hierarchical collection of DITA topics that can be processed to form an output. Maps do not contain the content of topics, but only references to them. These are known as topic references. Usually the maps are saved on disk or in a CMS with the extension '.ditamap'.

Maps can also contain relationship tables that establish relationships between the topics contained within the map. Relationship tables are also used to generate links in your published document.

You can use your map or bookmap to generate a deliverable using an output type such as XHTML, PDF, HTML Help or Eclipse Help.

## Bookmap

---

A bookmap is a specialized *ditamap* used for creating books. A bookmap supports book divisions such as chapters and book lists such as indexes.

# Index

## A

Archives 791, 792, 795  
 browse 792  
 edit 795  
 file browser 792  
 modify 792  
 Author Editing Mode 114  
 roles: content author, framework developer 114  
 Author Editor 113, 114, 117, 120, 121, 123, 124, 127, 129, 130, 131, 132, 134, 135, 147, 148, 149, 151, 152, 154, 155, 156, 158, 199  
 attributes view 121  
 breadcrumb 127  
     change tracking 124, 149, 151, 152, 154, 155, 156, 158  
     callouts 156  
     manage changes 151  
     managing comments 154, 155  
     the Review view 124, 158  
     track changes behavior 152  
     track changes limitations 154  
 content author role 117  
 contextual menu 132  
 edit content 135  
 editing XML 134  
 edit markup 134  
 elements view 120  
 entities view 123  
 external references 131  
 find/replace 132  
     navigation 127, 129  
     bookmarks 129  
     display the markup 129  
 outline view 117  
 position information tooltip 130  
 reload content 147  
 validation 147  
     whitespace handling 148, 199  
     versions differences 199  
 WYSIWYG editing 114  
 Author Settings 524, 525, 526, 527, 528, 529, 530, 531, 541, 544, 565, 569, 570, 576, 578, 580, 583, 585, 587  
     actions 524, 525, 526  
     insert section 525  
     insert table 526  
 Author default operations 531  
     content 530  
     configuring the content completion 530  
     content completion customization wizard 530  
     Java API 541, 544, 565, 569, 570, 576, 578, 580, 583, 585, 587  
 Author extension state listener 569  
 Author schema aware editing handler 570  
 configure XML node renderer customizer 587  
 CSS styles filter 578  
 customize outline icons 587  
 customize XML node 587  
 extensions bundle 565

## Author Settings (*continued*)

Java API (*continued*)  
     generate unique ID 587  
     references resolver 576  
     table cell row and column separators provider 585  
     table cell span provider 583  
     table column width provider 580  
 Java API example 541  
     menus 524, 527, 528  
     contextual menu 528  
     main menu 527  
     toolbars 524, 529  
     configure toolbar 529

## B

Bidirectional text 107, 113, 167  
 Author Mode 167  
 Grid Mode 113  
 Text Mode 107  
 bookmap 438  
 creating a bookmap 438

## C

Change color theme 1004  
 Change interface colors 1004  
 command line 743  
 Common Problems 1098  
 Comparing and Merging Documents 963, 964, 966, 967, 969, 971, 973, 974  
     directories comparison 964, 966, 967  
     compare images view 967  
     comparison results 966  
     user interface 964  
     files comparison 969, 971, 973, 974  
     character comparison 974  
     compare toolbar 971  
     file contents panel 973  
     files selector 973  
     main menu 969  
     word comparison 974  
 files comparison compare files fragments 967  
 Configuration 1045  
 CSS validator 1045  
 Configure Application 1041  
     Editor preferences 1041  
     spell check 1041  
 Configure the Application 548, 557, 1002, 1004, 1005, 1015, 1016, 1017, 1018, 1019, 1021, 1023, 1024, 1028, 1029, 1030, 1031, 1032, 1035, 1037, 1041, 1043, 1045, 1046, 1048, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1060, 1062, 1063, 1065, 1067, 1068, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1090  
 (S)FTP 1082  
 archive 1074  
 BOM 1015

Configure the Application (*continued*)

byte order mark 1015  
 certificates 1063  
 character encoding 1015  
 configure toolbars 1090  
 customize default options 1085  
 custom validation 1043  
     data sources 1065, 1067, 1068  
     download links for database drivers 1067  
     table filters 1068  
 Diff 1072  
 Diff appearance 1073  
 Diff directories 1073  
 Diff directories appearance 1074  
 document type association 1005  
 editor preferences 1016  
     Editor preferences 1017, 1018, 1019, 1021, 1023, 1024, 1028,  
       1029, 1030, 1031, 1032, 1035, 1037, 1041, 1043  
     author 1019  
     author profiling conditional text 1024  
     author track changes 1023  
     callouts 1024  
     code templates 1037  
     colors 1035  
     content completion 1032  
     document checking 1043  
     document templates 1041  
     elements and attributes by prefix 1035  
     format 1029  
     format - CSS 1031  
     format - JavaScript 1031  
     format - XML 1030  
     grid 1019  
     open/save 1035  
     pages 1017  
     print 1017  
     save hooks 1037  
     schema aware 1021  
     schema design 1028  
     text 1018  
     text/diagram 1018  
 editor variables 557, 1086  
 encoding 1015  
 external tools 1076  
 file types 1078  
 fonts 1004  
 global 1002  
 HTTP(S)/WebDAV preferences 1081  
     import 1062  
       date/time patterns 1062  
 import/export global options 1084  
 internationalization 548  
 menu shortcut keys 1077  
 messages 1071, 1083  
 outline 1083  
 perspectives layout 1015  
 plugins 1075  
 proxy preferences 1080  
 reset global options 1085  
 scenarios management 1086

Configure the Application (*continued*)

sharing preferences 1084  
     global options 1084  
     project options 1084  
 SSH 1083  
 SVN 1068  
 SVN Diff 1070  
 UTF-8 1015  
 views 1083  
 working copy 1070  
 XML 1045  
 XML catalog 1045  
 XML instances generator 1048  
 XML parser 1046  
 XProc engines 1048  
 XSLT 1050  
 XSLT/FO/XQuery 1050  
     XSLT/FO/XQuery preferences 1050, 1051, 1052, 1053, 1054,  
       1055, 1056, 1057, 1060  
     custom engines 1060  
     debugger 1057  
     FO Processors 1057  
     MSXML 1053  
     MSXML.NET 1054  
     profiler 1057  
     Saxon6 1050  
     Saxon HE/PE/EE 1051, 1055  
     Saxon-HE/PE/EE 1051  
     Saxon HE/PE/EE advanced options 1052, 1056  
     XPath 1060  
     XQuery 1055  
     XSLTProc 1053  
 Configure the interface 1004  
 Content Management System 855  
 Content Reuse 460, 461  
 content references 460  
 insert a direct content reference 461  
 reusable components 460  
 Copy/Paste 111, 166  
 grid editor 111  
 smart paste 166  
 CSS Support 595  
     CSS 2.1 features 595  
     supported selectors 595  
 CSS Support in Author 594, 599, 607  
     CSS 2.1 features 599  
     properties support table 599  
     Oxygen CSS extensions 594, 607  
     media type oxygen 594  
 Customization Support 518, 519, 523, 549, 556, 561, 562, 564,  
     572, 641  
     document type associations (advanced customization tutorial)  
       523, 549, 556, 561, 562, 564, 572  
     Author settings 523  
     basic association 549  
     configuring extensions - link target reference finder 572  
     configuring transformation scenarios 562  
     configuring validation scenarios 564  
     new file templates 556  
     XML Catalogs 561  
     example files 641  
     the Simple Documentation Framework Files 641

Customization Support (*continued*)  
 simple customization tutorial 518, 519  
 CSS 519  
 XML Schema 518

**D**

Databases 761, 797, 798, 817, 818, 836, 837, 838, 839, 840, 861  
 debugging with MarkLogic 839  
 limitations of the MarkLogic debugger 839  
 native XML databases (NXD) 817  
 Native XML databases (NXD) 818  
 Relational databases 798  
 SharePoint connection 861  
 WebDAV connection 840  
   XQuery 761, 836, 837, 838  
   debugging 838  
   drag and drop from the Data Source Explorer 836  
   transformation 837  
   validation 761  
 Debugging XSLT/XQuery Documents 768, 772, 781, 784  
 Java extensions 784  
   layout 768, 772, 781  
   information views 772  
   multiple output documents in XSLT 2.0 781  
 XSLT/XQuery debugger 781  
 Debugging XSLT / XQuery Documents 769  
   layout 769  
   Control toolbar 769  
 Develop an oXygen Plugin 984, 996  
 example - UppercasePlugin 996  
 introduction 984  
 Digital Signature 974, 976, 977, 978  
 canonicalizing files 976  
 certificates 977  
 signing files 977  
 verifying the signature 978  
 DITA MAP document type 497, 498  
 association rules 497  
   Author extension 498  
   catalogs 498  
 schema 497  
 DITA MAP Document Type 497, 498, 504  
   Author extension 498, 504  
   templates 504  
   transformation scenarios 498  
 DITA Maps 434, 438, 439, 440, 441, 443, 444, 453, 454, 457, 458, 459, 1106  
 creating a DITA Map 438  
 creating a topic 439  
   DITA OT customization support 453, 454  
   customizing Ant 453  
   increase the memory for Ant 453  
   resolve topic reference through an XML catalog 454  
   use your own custom build file 453  
 DITA OT installation plugin 457  
   DITA specialization 459  
   editing DITA Map specialization 459  
   DITA specialization support 458, 459  
   editing DITA Topic specialization 459  
 edit properties in DITA maps 444  
 inserting a reference 441

DITA Maps (*continued*)  
 inserting a topic group 444  
 inserting a topic heading 443  
 insert references 441  
 organizing topics 439  
 relationships between topics 439  
   transforming DITA Maps 444, 1106  
   running an ANT transformation 1106  
 validating a DITA Map 440  
 DITA Topics document type 488, 489  
 association rules 488  
   Author extensions 489  
   catalogs 489  
 schema 488  
 DITA Topics Document Type 488, 489, 496, 497  
   Author extensions 489, 496, 497  
   templates 497  
   transformation scenarios 496  
 DocBook Targetset document type 516  
 association rules 516  
 schema 516  
 DocBook Targetset Document Type 516  
 DocBook V4 document type 471, 478  
 association rules 471  
   Author extensions 471, 478  
   catalogs 471  
   templates 478  
 schema 471  
 DocBook V4 Document Type 471, 475  
   Author extensions 471, 475  
   transformation scenarios 475  
 DocBook V5 document type 481, 485  
 association rules 481  
   Author extensions 481, 485  
   catalogs 481  
   templates 485  
 schema 481  
 DocBook V5 Document Type 481, 482  
   Author extensions 481, 482  
   transformation scenarios 482  
 Documentum (CMS) Support 856, 857, 858, 859  
   actions 857, 858, 859  
   cabinets/folders 858  
   connection 858  
   resources 859  
 configuring a Documentum (CMS) data source 856, 857

**E**

Edit 98, 101, 102, 160, 171, 172, 173, 174, 178, 180, 186, 191, 192, 424, 427, 431, 795, 980, 1094  
 archives 795  
 associating a file extension 431  
 change the user interface language 1094  
 change user interface language 1094  
 Character map 173  
 check spelling 424  
 check spelling in files 427  
 close documents 191  
 conditional text 160  
 copy/paste 98  
 create new documents 174

*Edit (continued)*

- edit documents with long lines 431
- file properties 192
- find/replace 98
- find/replace (keyboard shortcuts) 102
- integrate external tools 980
- interface localization file 1094
- localize the user interface 1094
- open and close documents 174
- open documents 178
- open read-only files 431
- open remote documents (FTP/SFTP/WebDAV) 186
- open resource 180
- open the currently document in the system application 191
- Quick Find toolbar 101
- save documents 178
- scratch buffer 431
- Unicode documents 172
- Unicode support 172
- Unicode toolbar 173
- Editing Ant Build Files 294
- Editing CSS Stylesheets 380, 381, 382, 383
- Content Completion Assistant 381
- folding 382
  - format and indent (pretty print) 382
  - other editing actions 383
- Outline view 381
- validation 380
- Editing JavaScript Documents 404
- Editing JavaScript Files 404, 406, 407
- Content Completion Assistant 406
- Outline view 406
- Text mode 404
- validating JavaScript files 407
- Editing JSON Documents 399, 400, 401, 402
- convert XML to JSON 402
- folding 400
- Grid mode 401
- Outline view 402
- syntax highlight 400
- Text mode 399
- Validating JSON Documents 402
- Editing NVDL Schemas 395, 396, 397, 398
- Component Dependencies view 398
- editor specific actions 397
  - schema diagram 395, 396, 397
  - actions in the diagram view 396
  - full model view 395
  - Outline view 397
- searching and refactoring actions 397
- Editing RelaxNG Schemas 393
- Component Dependencies View 393
- Editing Relax NG Schemas 384, 385, 386, 387, 388, 389, 390
- editor specific actions 389
- Resource Hierarchy/Dependencies View 390
  - schema diagram 385, 386, 387, 388
  - actions 387
  - full model view 385
  - logical model view 386
  - Outline view 388
  - symbols 386
- searching and refactoring actions 389

Editing Schematron Documents 413

- searching and refactoring operations 413
- Editing Schematron Schemas 408, 409, 411
- contextual editing 411
- validation against Schematron 409
- Editing StratML Documents 403
- Editing SVG Documents 422, 423
- preview result pane 423
- standalone SVG viewer 423
- Editing WSDL Document 377
  - SOAP request 377
  - composing a SOAP request 377
- Editing WSDL Documents 362, 365, 366, 367, 368, 369, 372, 373, 374, 377, 379
- Component Dependencies view 372
- component occurrences 373
- composing web service calls with WSDL SOAP analyzer 377
- content completion 365
- contextual editing 366
- generate documentation for WSDL documents 374
- generate documentation for WSDL documents from command line 377
- generate documentation for WSDL documents in a custom format 377
- Outline view 362
- Quick Assist 373
- Resource Hierarchy/Dependencies view 369
- searching and refactoring operations 367
- searching and refactoring operations scope 368
  - SOAP request 379
  - testing remote WSDL files 379
  - UDDI registry browser 379
- Editing XML Documents 135, 193, 198, 199, 203, 204, 205, 207, 210, 211, 212, 213, 215, 216, 217, 218, 220, 223, 224, 225, 226, 228, 229, 230, 231, 233, 235, 237, 245, 246, 247, 249, 250, 251
- against a schema 216
  - associate a schema to a document 203, 204, 205, 207
  - add schema association in XML instance 205
  - learning a document structure 207
  - setting a default schema 204
  - supported schema types 204
- checking XML well-formedness 215
- code templates 135, 213
- content completion 135, 213
- converting between schema languages 235
  - document navigation 224, 225, 226, 228
  - bookmarks 224
  - fast navigation in Text mode 228
  - folding 225
  - navigation buttons 228
  - outline view 226
  - editor specific actions 247, 249, 250, 251
  - document actions 249
  - edit actions 247
  - refactoring actions 249
  - select actions 247
  - smart editing 250
  - source actions 247
  - split actions 247
  - syntax highlight depending on namespace prefix 251

**Editing XML Documents (*continued*)**

- grouping documents in XML projects 193, 198, 199, 229
- large documents 229
- new project 193
- project level settings 199
- project view 193
- team collaboration - Subversion 198
- image preview 245
- including document parts with XInclude 230
- locking and unlocking XML markup 246
- making a persistent copy of results 245
- markup transparency 246
- Resource Hierarchy/Dependencies view 233
- status information 245
  - streamline with content completion 207, 210, 211, 212, 213
  - the Annotation panel 211
  - the Attributes view 212
  - the Elements view 212
  - the Entities view 213
  - the Model panel 210
  - validation against a schema 216, 217, 218, 220, 223, 224
  - automatic validation 218
  - custom validation 218
  - marking validation errors 216
  - references to XML Schema specification 224
  - resolving references to remote schemas with an XML Catalog 224
  - validation actions 223
  - validation example 217
  - validation scenario 220
- working with XML Catalogs 231
- XML tree nodes 237
- Editing XML Schemas** 301, 336, 337, 339, 342, 344, 347, 348, 351, 354, 355, 357
- Component Dependencies** view 337
- contextual editing 336
  - generate documentation for XML Schema 342, 344, 347, 348 as HTML 344
  - as PDF, DocBook or custom format 347
  - from command line 348
- relational database table to XML schema 357
- Resource Hierarchy/Dependencies view 339
  - schema instance generator 351, 354
    - running from command line 354
- schema regular expressions builder 355
- searching and refactoring actions 336
- Editing XProc Scripts** 407
- Editing XQuery Documents** 359, 360, 361
- folding 360
  - generate HTML documentation 361
- Editing XSL Stylesheets** 288, 289, 292
- Component Dependencies** view 288
- quick assist support 289
- XSpec 292
  - Editing XSLT Schemas** 265
  - contextual editing 265
- Editing XSLT Stylesheets** 264, 265, 266, 267, 270, 272, 274, 275, 278, 280, 281, 282, 283, 285
  - content completion 266, 267
  - in XPath expressions 267
- find XSLT references and declarations 282
- generate documentation for XSLT stylesheets 275

**Editing XSLT Stylesheets (*continued*)**

- generate documentation for XSLT Stylesheets 278, 280, 281
  - as HTML 278
  - from command line 281
  - in custom format 280
- Outline view 272
- refactoring actions 283
- Resource Hierarchy/Dependencies** view 285
  - validation 265
  - custom validation 265
  - validation scenario 265
- XSLT Input** view 270
- XSLT stylesheet** documentation 274
- Edit Large Documents** 430
  - larger than 300 MB 430
  - smaller than 300 megabytes 430
- EPUB Document Type** 515
- Extend Oxygen with plugins 984
  - implement plugin 984
  - Extend Oxygen with Plugins 983, 985, 987, 988, 989, 991, 992, 995
    - implement plugin 985, 987, 988, 989, 991, 992, 995
    - CMS integration plugin 992
    - components validation plugin 987
    - custom protocol plugin 988, 989, 991
    - document plugin 992
    - general plugin 991
    - how to install a plugin 985
    - how to write a custom protocol plugin 995
    - resource locking custom protocol plugin 989
    - selection plugin 991
  - external process 743

**F**

- Find/Replace** 98, 100, 101, 102, 132
- Author editor** 132
- Find All Elements/Attributes** dialog box 100
- keyboard shortcuts 102
- Quick Find** toolbar 101
- Format and indent 237
- Format and Indent 1031

**G**

- Getting Started** 85, 86, 89, 90, 91, 963
- dockable views and editors** 91
  - perspectives 86, 89, 90, 963
  - database 90
  - editor 86
  - tree editor 963
  - XQuery debugger 90
  - XSLT debugger 89
- grid editor** 110
  - navigation 110
  - collapse all 110
  - collapse children 110
  - collapse others 110
  - expand all 110
  - expand children 110
- Grid Editor** 108, 109, 110, 111
- add nodes 111

**Grid Editor (*continued*)**

clear column content 111  
 copy/paste 111  
 drag and drop 111  
 duplicate nodes 111  
 inserting table column 110  
 insert table row 110  
 layouts (grid and tree) 109  
 navigation 109  
 refresh layout 111  
 sort table column 110  
 start and stop editing a cell value 111

**I**

Importing data 846  
**Importing Data** 846, 848, 850  
 from a database 846  
 table content as XML document 846  
 from database 848  
 convert table structure to XML Schema 848  
 from HTML files 850  
 from MS Excel 848  
 from text files 850  
 Installation 27, 28, 29, 30, 31, 32, 33, 34  
 all platforms version 28, 29, 31, 32, 34  
 Linux 30, 33  
 multiple instances (Unix / Linux server) 34  
 OS X installation 29  
 unattended (Windows and Linux only) 27, 31  
 Windows installation 27, 32  
 Windows terminal server 32  
 Integrating the Ant tool 981

**J**

**JATS NISO Journal Article Tag Suite Document Type** 514

**L**

License 36, 37, 38, 39, 40, 43, 44, 45  
 floating (concurrent) license 38  
 floating license server 43  
 floating license servlet 40  
 license server installed on OS X Linux Unix 44  
 multiple named-user licenses 38  
 named-user license 37  
 register a license key 36  
 release floating license 39  
 releasing a license key 45  
 transferring a license key 45  
 unregistering a license key 45

**M**

Master Files 200

**N**

**Native XML Databases (NXD)** 809, 817, 818, 819, 820, 821, 822  
 database connections configuration 818, 819, 821  
 Berkeley DB XML 818  
 Documentum xDb (X-Hive/DB) 821  
 eXist 819  
 data sources configuration 817, 818, 819, 820, 821  
 Berkeley DB XML 818  
 Documentum xDb (X-Hive/DB) 821  
 eXist 819  
 MarkLogic 820  
 resource management 809, 822  
 Data Source Explorer view 809, 822

**O**

**OutOfMemory** 1029, 1035, 1057, 1095, 1096, 1098  
**Out Of Memory** 978, 1029, 1035, 1057, 1095, 1096, 1098, 1099  
 large documents error 1099  
 Large File Viewer 978  
 opening XML documents closed a long time ago 1098  
**OutOfMemoryError** 1029, 1035, 1057, 1095, 1096, 1098  
 outside oxygen 743  
**Oxygen CSS Extensions** 597, 599, 604, 607, 609, 610, 611, 612, 613  
 additional properties 609, 610, 611, 612  
 display tags 612  
 editable property 611  
 folding elements 609  
 link elements 611  
 morph value 611  
 placeholders for empty elements 610  
**oXygen CSS custom functions** 613  
 supported features from CSS level 3 597, 604, 607  
 additional custom selectors 607  
**attr( ) function** 604  
 namespace selectors 597  
 supported features from CSS level 4 599  
 subject selectors 599

**P**

**Performance Problems** 1098  
 external processes 1098  
 large documents 1098  
 problems on Linux/Solaris 1098  
 Preferences 1002  
 Pretty print 237  
**Profile DITA step by step** 466  
 conditional text 466  
 profiling tutorial 466  
**Profiling** 463, 464, 788  
 conditional text 464  
 filter content 464  
 filter content 463  
 conditional text 463  
**XSLT stylesheets and XQuery documents** 788  
**Profiling XSLT Stylesheets and XQuery Documents** 788, 789  
 profiling information 788, 789  
 Hotspots view 789  
 Invocation tree view 788

Profiling XSLT Stylesheets and XQuery Documents (*continued*)  
 XSLT/XQuery profiler 789

## Q

Querying Documents 359, 752, 753, 757, 758, 760, 761, 762  
 running XPath and XQuery expressions 753  
 XPath/XQuery Builder view 753  
 running XPath expressions 752  
 XPath toolbar 752  
 XQuery 359, 757, 758, 760, 761, 762  
 Input view 760  
 other editing actions 762  
 Outline view 359, 758  
 syntax highlight and content completion 757  
 transforming XML documents; advanced Saxon B/SA options  
 762  
 validation 761

## R

Relational Databases 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 813, 815, 817, 822, 846, 848  
 connections configuration 798, 799, 801, 803, 804, 806, 808  
 generic JDBC 803  
 IBM DB2 connection 799  
 JDBC-ODBC connection 808  
 Microsoft SQL Server 801  
 MySQL 804  
 Oracle 11g 806  
 PostgreSQL 8.3 808  
 creating XML Schema from databases 848  
 data sources configuration 798, 800, 802, 803, 805, 807  
 generic JDBC data source 802  
 IBM DB2 798  
 Microsoft SQL Server 800  
 MySQL 803  
 Oracle 11g 805  
 PostgreSQL 8.3 807  
 importing from databases 846  
 resource management 809, 813, 822  
 Data Source Explorer view 809, 822  
 Table Explorer view 813  
 SQL execution support 815, 817  
 drag and drop from the Data Source Explorer 815  
 executing SQL statements 817  
 SQL validation 817  
 Relax NG Schema Editor 384  
 contextual editing 384

## S

Schematron Quick Fix 252  
 SharePoint Connection 861, 864, 865  
 actions at connection level 864  
 actions at file level 865  
 actions at folder level 864  
 configuration 861  
 SQF 252  
 Startup parameter 1095, 1096  
 application launchers parameters 1095  
 command line scripts parameters 1096

Subject scheme 438  
 creating a Subject scheme 438  
 SVN Branches/Tags 901, 903, 905, 906, 907, 909, 913, 914  
 create a branch/tag 901  
 merging 903, 905, 906, 907, 909  
 merge changes 905  
 merge two different trees 909  
 reintegrate a branch 907  
 reverse merge 905  
 synchronize branch 906  
 relocate a working copy 914  
 switch the repository location 913  
 SVN Client 868, 876, 877, 878, 880, 881, 884, 889, 899, 900, 901, 915, 926, 927, 928, 929, 930, 939, 944, 945, 946, 947, 949, 952, 953, 954, 955, 956, 959  
 Annotations view 949  
 Compare view 952, 953, 954  
 Compare images view 954  
 toolbar 953  
 Console view 955  
 define a repository location 878  
 add/edit/remove repository locations 878  
 authentication 878  
 define a working copy 881  
 Help view 955  
 History view 944, 945, 946  
 history filter dialog 945  
 history filter field 946  
 image preview 954  
 main window 868, 876, 877  
 main menu 868  
 status bar 877  
 toolbar 876  
 views 868  
 obtain information regarding a resource 899, 900  
 request history 900  
 request status information 899  
 Preferences 959  
 Properties view 954, 955  
 toolbar and contextual menu 955  
 Repositories view 929, 930  
 contextual menu actions 930  
 toolbar 930  
 Resource History view 947  
 Directory Change Set view 947  
 popup menu on double selection 947  
 Revision Graph 956  
 share a project 880  
 sparse checkouts 928  
 SVN branches/tags 901, 915  
 patch 915  
 SVN properties 900, 955  
 Add / Edit / Remove 955  
 SVN working copy resources 884  
 synchronize with the SVN repository 889  
 use an existing working copy 884  
 Working Copy view 939  
 contextual menu actions 939  
 working with repositories 926, 927  
 copy/move/delete resources 927  
 export resources 926

SVN working copy resources 888

- lock / unlock resources 888

- locked items 888

SVN Working Copy Resources 884, 886, 887, 888, 889

add resources to version control 884

copy resources 887

delete resources 886

edit files 884

ignore resources 886

- lock/unlock resources 887, 888, 889

- locking a file 888

- scanning for locks 888

- unlocking a file 889

move resources 887

rename resources 887

Synchronize with the SVN Repository 889, 891, 892, 893, 894,

- 895, 896, 899

commit changes 896

integration with Bug Tracking tools 899

- resolve conflicts 891, 892, 893, 894, 895

- content conflicts vs property conflicts 891

- drop incoming modifications 895

- edit real content conflicts 892

- merge conflicted resources 894

- real conflicts vs mergeable conflicts 891

- revert changes 893

- tree conflicts 895

update the working copy 896

view differences 889

Synchronize With The SVN Repository 889

## T

TEI ODD document type 507, 508

association rules 507

- Author extensions 508

- catalogs 508

schema 507

TEI ODD Document Type 507, 508, 509

- Author extensions 508, 509

- templates 509

- transformation scenarios 509

TEI P4 document type 509

association rules 509

- Author extensions 509

- catalogs 509

schema 509

TEI P4 Document Type 509, 511

- Author extensions 509, 511

- templates 511

- transformation scenarios 511

TEI P5 document type 511, 512

association rules 511

- Author extensions 512

- catalogs 512

schema 511

TEI P5 Document Type 511, 512, 513

- Author extensions 512, 513

- templates 513

- transformation scenarios 513

Text Editing Mode 98, 102, 105, 106, 191

change the font size 105

Text Editing Mode (*continued*)

drag and drop 106

find and replace text in multiple files 102

insert file at caret position 106

open file at caret position 106

open file at caret position in system application 106

print a file 106

switch between opened tabs 191

undo and redo 98

word/line editor actions 105

Tools 867

Transformation Scenario 690, 691, 692, 693, 694, 696, 719, 720

built-in transformation scenarios 720

- new transformation scenario 690, 691, 692, 693, 694, 696, 719

- additional XSLT stylesheets 694

- configure transformation scenario 690

- create a transformation scenario 719

- XML transformation with XSLT 691

- XQuery parameters 692

- XSLT/XQuery extensions 693, 696

- XSLT parameters 692

sharing the transformation scenarios; project level scenarios 720

Transforming Documents 689, 690, 721, 724, 726, 727, 730

custom XSLT processors 726

output formats 730

supported XSLT processors 724

transformation scenario 690

Transformation Scenarios view 721

XSL-FO processors 727

XSLT processors extensions paths 726

## U

Unicode 1031

Uninstalling the application 47

Upgrade 45

check for new version 45

## V

Validating XML Documents 215

Validation Scenario 222

sharing the validation scenarios; project level scenarios 222

## W

WebDAV Connection 840, 841

actions at connection level 840

actions at file level 841

actions at folder level 841

configuration 840

WebHelp 743

WebHelp Internationalization 501

- WebHelp localization 501

- WebHelp i18n 501

whitespace 1031

Whitespace handling 237

Workspace Access 986

**X**

XHTML document type 505  
 association rules 505  
     Author extensions 505  
     catalogs 505  
 CSS 505  
 schema 505  
 XHTML Document Type 504, 505, 507  
     Author extensions 505, 507  
     templates 507  
     transformation scenarios 507  
 XML Outline View 118, 119, 226, 227, 228  
     Author 119  
     outline filters 119  
 contextual menu 119  
     document structure change 119, 227  
     contextual menu 227  
 document tag selection 228  
 modification follow-up 118, 227  
 outline filters 227  
 XML document overview 118, 226  
 XML Quick Fixes 252  
 XML Schema Diagram Editor 301, 303, 304, 307, 308, 309, 310,  
     312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322,  
     329, 330, 332, 333  
 Attributes view 330  
 editing actions 322  
 edit schema namespaces 333  
     Facets view 332, 333  
     editing patterns 333  
     group schema components 320  
     attributes 320  
     constraints 320  
     substitutions 320  
 navigation 321  
 Outline view 329  
     schema components 303, 304, 307, 308, 309, 310, 312, 313,  
         314, 315, 316, 317, 318, 319  
     xs:alternative 312  
     xs:any 315  
     xs:anyAttribute 316  
     xs:assert 318  
     xs:attribute 307  
     xs:attributeGroup 308  
     xs:complexType 309  
     xs:element 304  
     xs:field 318

XML Schema Diagram Editor (*continued*)  
     schema components (*continued*)  
         xs:group 313  
         xs:import 313  
         xs:include 313  
         xs:key 317  
         xs:keyRef 317  
         xs:notation 314  
         xs:openContent 319  
         xs:override 314  
         xs:redefine 314  
         xs:schema 303  
         xs:selector 318  
         xs:sequence, xs:choice, xs:all 315  
         xs:simpleType 310  
         xs:unique 317  
     the Palette view 333  
     validation 321  
 XML Schema Text Editor 334, 335, 349  
 content completion 334  
 flatten an XML Schema 349  
 references to XML Schema specification 335  
 XML Schema actions 335  
 XML serialization 237  
 XQJ Connection 763  
 XQJ configuration 763  
 XQJ Support 763  
 XQJ processor configuration 763  
 XSLT/XQuery Debugger 772, 773, 774, 775, 776, 777, 778, 779,  
     781, 782  
 debug steps 781  
 determining what XSLT/XQuery expression generated particular  
     output 782  
     using breakpoints 782  
     inserting breakpoints 782  
     removing breakpoints 782  
     viewing processing information 772, 773, 774, 775, 776, 777,  
         778, 779  
     breakpoints view 773  
     context node view 772  
     messages view 774  
     node set view 779  
     output mapping stack view 776  
     stack view 775  
     templates view 778  
     trace history view 777  
     variables view 779  
     XPath watch view 773

