

Oxygen XML Editor Eclipse Plugin 15.2

Notice

Copyright

Oxygen XML Editor plugin User Manual

Syncro Soft SRL.

Copyright © 2002-2014 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 book, and SyncRO Soft SRL was aware of a trademark claim, the designations have been printed in caps or initial caps. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

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 End User License Agreement, as well as information regarding support for this product, while under warranty, is available through the [Oxygen XML Web site](#).

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 Web site](#).

Downloading documents. For the most current versions of documentation, see the [Oxygen XML Support Web site](#).

Contact Syncro Soft SRL. Syncro Soft SRL provides a telephone number and e-mail address for you to report problems or to ask questions about your product, see the [Oxygen XML Web site](#).

Contents

Chapter 1: Introduction.....	25
Key Features and Benefits of Oxygen XML Editor plugin	26
Chapter 2: Installation.....	29
Installation Requirements.....	30
Platform Requirements.....	30
Operating System.....	30
Environment Requirements.....	30
Tools.....	30
Java Virtual Machine Prerequisites.....	30
Installation Instructions.....	31
Eclipse 3.4 - 4.2 Plugin Installation - The Update Site Method.....	31
Eclipse 3.4 - 4.2 Plugin Installation - The Zip Archive Method.....	31
Making Oxygen XML Editor plugin Portable.....	31
Obtaining and Registering a License Key.....	32
Named User License Registration.....	33
Named User License Registration with Text File.....	34
Named User License Registration with XML File.....	34
How Floating (Concurrent) Licenses Work.....	34
Setting up a Floating License Server Running as a Java Servlet.....	34
Setting up a Floating License Server Running as a Standalone Process.....	36
Request a Floating License from a License Server Running as a Standalone Process.....	37
Request a Floating License from a License Server Running as a Java Servlet.....	38
Release a Floating License.....	39
License Registration with an Activation Code.....	39
Unregistering the License Key.....	39
Upgrading Oxygen XML Editor plugin.....	39
Upgrading the Eclipse Plugin.....	40
Checking for New Versions.....	40
Uninstalling the Oxygen XML Editor plugin.....	40
Uninstalling the Eclipse plugin.....	40
Performance Problems.....	41
External Processes.....	41
Chapter 3: Getting Started.....	43
Getting Help.....	44
Supported Types of Documents.....	44
Perspectives.....	44

Oxygen XML Editor plugin XML Perspective	45
The Oxygen XML Editor plugin Custom Menu.....	45
The Oxygen XML Editor plugin Toolbar Buttons.....	45
The Editor Pane.....	46
The Outline View.....	46
The Oxygen XML Editor plugin Text View.....	46
The Oxygen XML Editor plugin Browser View.....	47
The Results View.....	47
The Oxygen XML Editor plugin XPath Results View.....	48
Supported Editor Types.....	48
XSLT Debugger Perspective	49
XQuery Debugger Perspective	51
Oxygen XML Editor plugin Database Perspective	52

Chapter 4: Editing Modes.....55

Text Editing Mode.....	56
Finding and Replacing Text in the Current File.....	56
The Find All Elements / Attributes Dialog.....	56
Grid Editing Mode.....	57
Layouts: Grid and Tree.....	58
Grid Move Navigation.....	58
Specific Grid Actions.....	59
Sorting a Table Column.....	59
Inserting a Row in a Table.....	59
Inserting a Column in a Table.....	59
Clearing the Content of a Column.....	59
Adding Nodes.....	60
Duplicating Nodes.....	60
Refresh Layout.....	60
Start Editing a Cell Value.....	60
Stop Editing a Cell Value.....	60
Drag and Drop in the Grid Editor.....	60
Copy and Paste in the Grid Editor.....	60
Bidirectional Text Support in Grid Mode.....	62
Author Editing Mode.....	62
Tagless XML Authoring.....	63
General Author Presentation.....	64
Author Views.....	65
The Author Editor.....	71
Managing Changes.....	90
Review.....	94
Profiling / Conditional Text.....	101
Smart Paste Support.....	106
Bidirectional Text Support in Author Mode.....	106

Controlling the Text Direction Using XML Markup.....	106
Controlling the Text Direction Using the Unicode Direction Formatting Codes.....	108

Chapter 5: Editing Documents.....109

Working with Unicode.....	110
Opening and Saving Unicode Documents.....	110
Creating, Opening, and Closing Documents.....	110
Creating Documents.....	110
Oxygen XML Editor plugin Plugin Wizard for New Document	111
Creating Documents Based on Templates.....	114
Saving Documents.....	115
Opening and Saving Remote Documents via FTP/SFTP	115
The Open Using FTP/SFTP Dialog.....	115
Changing File Permissions on a Remote FTP Server.....	117
WebDAV over HTTPS.....	117
Opening the Current Document in System Application.....	119
Closing Documents.....	119
The Contextual Menu of the Editor Tab.....	119
Viewing File Properties.....	119
Grouping Documents in XML Projects.....	120
Creating a New Project.....	120
Create an Oxygen XML Editor plugin XML Project.....	121
Moving/Renaming Resources in the Navigator View.....	121
Defining Master Files at Project Level.....	122
Introduction.....	122
Master Files Benefits.....	123
Enabling the Master Files Support.....	123
Detecting Master Files.....	123
Adding/Removing a Master File.....	124
Editing XML Documents.....	124
Associate a Schema to a Document.....	124
Setting a Schema for Content Completion.....	124
Learning Document Structure.....	127
Streamline with Content Completion.....	127
Set Schema for Content Completion.....	129
Content Completion in Documents with Relax NG Schemas.....	129
Schema Annotations.....	130
Content Completion Helper Views.....	131
Code Templates.....	134
Validating XML Documents.....	136
Checking XML Well-Formedness.....	136
Validating XML Documents Against a Schema.....	137
Document Navigation.....	144
Folding of the XML Elements.....	144

Outline View.....	145
Large Documents.....	148
Including Document Parts with DTD Entities.....	148
Including Document Parts with XInclude.....	148
Working with XML Catalogs.....	150
Resolve Schemas Through XML Catalogs	151
XML Resource Hierarchy/Dependencies View.....	151
Moving/Renaming XML Resources.....	153
Converting Between Schema Languages.....	153
Formatting and Indenting Documents (Pretty Print).....	155
How to use zero size indent.....	156
Editing Modular XML Files in the Master Files Context.....	156
Managing ID/IDREFS.....	157
Highlight IDs Occurrences in Text Mode.....	157
Search and Refactor Actions for ID/IDREFS.....	157
Quick Assist Support for ID/IDREFS in Text Mode.....	158
Search and Refactor Operations Scope.....	158
Viewing Status Information.....	159
XML Editor Specific Actions.....	159
Edit Actions.....	159
Select Actions.....	160
Source Actions.....	160
XML Document Actions.....	160
XML Refactoring Actions.....	161
Smart Editing.....	162
Syntax Highlight Depending on Namespace Prefix.....	162
Editor Highlights.....	163
Batch Editing Actions on Highlights.....	163
Editing XHTML Documents.....	164
Editing XSLT Stylesheets.....	164
Validating XSLT Stylesheets.....	164
Custom Validation of XSLT Stylesheets.....	164
Associate a Validation Scenario.....	164
Editing XSLT Stylesheets in the Master Files Context.....	165
Syntax Highlight.....	165
Content Completion in XSLT Stylesheets.....	165
Content Completion in XPath Expressions.....	166
Code Templates.....	170
The XSLT/XQuery Input View.....	170
The XSLT Input View.....	170
The XSLT Outline View.....	171
XSLT Stylesheet Documentation Support.....	173
Generating Documentation for an XSLT Stylesheet.....	174
Generate Documentation in HTML Format.....	176
Generate Documentation in a Custom Format.....	179

Generating Documentation From the Command Line Interface.....	179
Finding XSLT References and Declarations.....	180
Highlight Component Occurrences.....	181
XSLT Refactoring Actions.....	181
XSLT Resource Hierarchy/Dependencies View.....	182
Moving/Renaming XSLT Resources.....	183
Component Dependencies View.....	184
XSLT Quick Assist Support.....	185
Linking Between Development and Authoring.....	185
XSLT Unit Test (XSpec).....	185
Editing XML Schemas.....	187
Editing XML Schema in the Master Files Context.....	187
XML Schema Text Editing Mode.....	187
Special Content Completion Features.....	187
Flatten an XML Schema.....	188
Highlight Component Occurrences.....	189
XML Schema Diagram Editing Mode.....	190
Introduction.....	190
XML Schema Components.....	191
Navigation in the Schema Diagram.....	210
Schema Validation.....	210
Schema Editing Actions.....	211
Schema Outline View.....	217
The Attributes View.....	218
The Facets View.....	219
The Palette View.....	220
Edit Schema Namespaces.....	221
Create an XML Schema From a Relational Database Table.....	221
Generate Sample XML Files.....	221
The Schema Tab.....	222
The Options Tab.....	222
The Advanced Tab.....	224
XML Schema Regular Expressions Builder.....	225
Generating Documentation for an XML Schema.....	226
Generate Documentation in HTML Format.....	228
Generate Documentation in PDF, DocBook or a Custom Format.....	231
Generating Documentation From the Command-Line Interface.....	232
Searching and Refactoring Actions.....	233
XML Schema Resource Hierarchy / Dependencies View.....	235
Moving/Renaming XML Schema Resources.....	237
Component Dependencies View.....	237
XML Schema Quick Assist Support.....	238
Linking Between Development and Authoring.....	239
XML Schema 1.1.....	239
Setting the XML Schema Version.....	240

Editing XQuery Documents.....	241
XQuery Outline View.....	241
Folding in XQuery Documents.....	242
Generating HTML Documentation for an XQuery Document.....	243
Editing WSDL Documents.....	244
WSDL Outline View.....	245
Content Completion in WSDL Documents.....	248
Editing WSDL Documents in the Master Files Context.....	248
Searching and Refactoring Operations in WSDL Documents.....	249
Searching and Refactoring Operations Scope in WSDL Documents.....	249
WSDL Resource Hierarchy/Dependencies View in WSDL Documents.....	250
Moving/Renaming WSDL Resources.....	251
Component Dependencies View in WSDL Documents.....	252
Highlight Component Occurrences in WSDL Documents.....	253
Quick Assist Support in WSDL Documents.....	253
Generating Documentation for WSDL Documents.....	254
Generating Documentation for WSDL Documents in HTML Format.....	256
Generating Documentation for WSDL Documents in a Custom Format.....	257
Generating Documentation for WSDL Documents from the Command Line.....	257
WSDL SOAP Analyzer.....	257
Composing a SOAP Request.....	257
Testing Remote WSDL Files.....	259
The UDDI Registry Browser.....	259
Editing CSS Stylesheets.....	260
Validating CSS Stylesheets.....	260
Specify Custom CSS Properties.....	260
Content Completion in CSS Stylesheets.....	260
CSS Outline View.....	261
Folding in CSS Stylesheets.....	262
Formatting and Indenting CSS Stylesheets (Pretty Print).....	262
Other CSS Editing Actions.....	262
Editing Relax NG Schemas.....	262
Editing Relax NG Schema in the Master Files Context.....	262
Relax NG Schema Diagram.....	262
Introduction.....	263
Full Model View.....	263
Symbols Used in the Schema Diagram.....	263
Logical Model View.....	264
Actions Available in the Diagram View.....	265
Relax NG Outline View.....	265
Relax NG Editor Specific Actions.....	266
Searching and Refactoring Actions.....	267
RNG Resource Hierarchy/Dependencies View.....	267
Moving/Renaming RNG Resources.....	269
Component Dependencies View.....	269

RNG Quick Assist Support.....	270
Configuring a Custom Datatype Library for a RELAX NG Schema.....	271
Linking Between Development and Authoring.....	271
Editing NVDL Schemas.....	271
NVDL Schema Diagram.....	271
Introduction.....	271
Full Model View.....	271
Actions Available in the Diagram View.....	272
NVDL Outline View.....	273
NVDL Editor Specific Actions.....	273
Searching and Refactoring Actions.....	273
Component Dependencies View.....	273
Linking Between Development and Authoring.....	274
Editing JSON Documents.....	274
JSON Editor Text Mode.....	274
Syntax highlight in JSON Documents.....	275
Folding in JSON.....	275
JSON Editor Grid Mode.....	276
JSON Outline View.....	277
Validating JSON Documents.....	277
Convert XML to JSON.....	277
Editing StratML Documents.....	278
Editing JavaScript Documents.....	279
JavaScript Editor Text Mode.....	279
Content Completion in JavaScript Files.....	280
JavaScript Outline View.....	281
Validating JavaScript Files.....	282
Editing XProc Scripts.....	282
Editing Schematron Schemas.....	283
Validate an XML Document.....	284
Validating Schematron Documents.....	284
Content Completion in Schematron Documents.....	285
Code Templates.....	285
RELAX NG/XML Schema with Embedded Schematron Rules.....	285
Editing Schematron Schema in the Master Files Context.....	286
Schematron Resource Hierarchy/Dependencies View.....	286
Moving/Renaming Schematron Resources.....	287
Highlight Component Occurrences in Schematron Documents.....	287
Searching and Refactoring Operations in Schematron Documents.....	288
Searching and Refactoring Operations Scope in Schematron Documents.....	288
Quick Assist Support in Schematron Documents.....	289
Spell Checking.....	289
Spell Checking Dictionaries.....	291
Dictionaries for the Hunspell Checker.....	291
Dictionaries for the Java Checker.....	291

Learned Words.....	292
Ignored Words.....	292
Automatic Spell Check.....	292
Spell Checking in Multiple Files.....	293
Handling Read-Only Files.....	294
Associating a File Extension with Oxygen XML Editor plugin.....	294
Terms.....	295

Chapter 6: Author for DITA.....297

Creating DITA Maps and Topics.....	298
Editing DITA Maps.....	298
Editing Actions.....	300
Creating a Map.....	301
Validating DITA Maps.....	301
Using a Root Map.....	303
Create a Topic in a Map.....	303
Organize Topics in a Map.....	303
Create a Bookmark.....	304
Create a Subject Scheme.....	304
Create Relationships Between Topics.....	304
Advanced Operations.....	305
Inserting a Reference, a Key Definition, a Topic Set.....	305
Inserting a Topic Heading.....	307
Inserting a Topic Group.....	307
Edit Properties.....	308
Transforming DITA Maps and Topics.....	308
Creating a DITA Transformation Scenario.....	308
Compiled HTML Help (CHM) Output Format.....	310
The <i>TocJS</i> Transformation.....	310
Kindle Output Format.....	311
Migrating OOXML Documents to DITA.....	311
Customizing a DITA Scenario.....	311
The Parameters Tab.....	311
The <i>Filters</i> Tab.....	312
The <i>Advanced</i> Tab.....	313
The Output Tab.....	315
The <i>FO Processor</i> Tab.....	315
Running a DITA Map ANT Transformation.....	317
Set a Font for PDF Output Generated with Apache FOP.....	317
How does the DITA Open Toolkit PDF font mapping work?.....	317
Tips and Tricks.....	317
Debugging PDF Transformations.....	317
The PDF Processing Fails to Use the DITA OT and Apache FOP.....	318
Topic References outside the main DITA Map folder.....	319

Embedding videos in the WebHelp output.....	319
Syntax Highlight Inside Codeblock Sections.....	319
DITA-OT Customization.....	320
Support for Transformation Customizations.....	320
Using Your Custom Build File.....	320
Customizing the Oxygen XML Editor plugin Ant Tool.....	320
Increasing the Memory for the Ant Process.....	320
Resolving Topic References Through an XML Catalog.....	321
DITA to PDF Output Customization.....	321
Header and Footer Customization.....	322
Installing a plugin in the DITA Open Toolkit.....	322
Creating a Simple DITA OT HTML and PDF Customization Plugin.....	323
DITA Specialization Support.....	324
Integration of a DITA Specialization.....	324
Editing DITA Map Specializations.....	325
Editing DITA Topic Specializations.....	325
Use a New DITA Open Toolkit in Oxygen XML Editor plugin.....	326
Reusing Content.....	326
Working with Content References.....	326
How to Work with Reusable Components.....	327
Insert a Direct Content Reference.....	328
The Insert Content Reference Dialog.....	328
Moving and Renaming Resources.....	329
DITA Profiling / Conditional Text.....	330
Profiling / Conditional Text Markers.....	331
Profiling with a Subject Scheme Map.....	332
Publish Profiled Text.....	332
How to Profile DITA Content.....	333
Working with MathML.....	333
MathML Equations in the HTML Output.....	333

Chapter 7: Predefined Document Types.....335

Document Type.....	336
The DocBook 4 Document Type.....	336
DocBook 4 Author Extensions.....	336
DocBook 4 Transformation Scenarios.....	339
WebHelp Output Format.....	339
WebHelp with Feedback Output Format.....	341
WebHelp Mobile Output Format.....	347
DocBook 4 Templates.....	347
Inserting olink Links in DocBook 5 Documents.....	348
The DocBook 5 Document Type.....	350
DocBook 5 Author Extensions.....	350
DocBook 5 Transformation Scenarios.....	351

DocBook to EPUB Transformation.....	351
WebHelp Output Format.....	351
WebHelp with Feedback Output Format.....	353
WebHelp Mobile Output format.....	360
DocBook to PDF Output Customization.....	360
DocBook 5 Templates.....	361
Inserting olink Links in DocBook 5 Documents.....	361
The DocBook Targetset Document Type.....	363
DocBook Targetset Templates.....	364
The DITA Topics Document Type.....	364
DITA Author Extensions.....	364
DITA Transformation Scenarios.....	371
DITA Templates.....	371
The DITA Map Document Type.....	372
DITA Map Author Extensions.....	372
DITA Map Transformation Scenarios.....	373
WebHelp Output Format.....	373
WebHelp with Feedback Output Format	377
Mobile WebHelp Output Format.....	382
DITA Map Templates.....	382
The XHTML Document Type.....	383
XHTML Author Extensions.....	383
XHTML Transformation Scenarios.....	384
XHTML Templates.....	385
The TEI ODD Document Type.....	385
TEI ODD Author Extensions.....	385
TEI ODD Transformation Scenarios.....	387
TEI ODD Templates.....	387
The TEI P4 Document Type.....	387
TEI P4 Author Extensions.....	388
TEI P4 Transformation Scenarios.....	389
TEI P4 Templates.....	390
Customization of TEI Frameworks Using the Latest Sources.....	390
The TEI P5 Document Type.....	390
TEI P5 Transformation Scenarios.....	391
TEI P5 Templates.....	391
Customization of TEI Frameworks Using the Latest Sources.....	391
Customization of TEI Frameworks Using the Compiled Sources.....	392
The EPUB Document Type.....	392
Chapter 8: Author Developer Guide.....	393
Simple Customization Tutorial.....	395
XML Schema.....	395
CSS Stylesheet.....	395

The XML Instance Template.....	398
Advanced Customization Tutorial - Document Type Associations.....	399
Document Type.....	399
Document Type Settings.....	399
Editing attributes in-place using form controls.....	419
Localizing Frameworks.....	419
How to Deploy a Plugin or a Framework as an Oxygen XML Editor plugin Add-on.....	420
Creating the Basic Association.....	420
First Step - XML Schema.....	421
Schema Settings.....	422
Second Step - The CSS.....	422
Defining the General Layout.....	422
Styling the <code>section</code> Element.....	423
Styling the Inline Elements.....	424
Styling Images.....	424
Testing the Document Type Association.....	425
Organizing the Framework Files.....	426
Packaging and Deploying.....	426
Configuring New File Templates.....	427
Create Your Own Stylesheet Templates.....	430
Configuring XML Catalogs.....	430
Configuring Transformation Scenarios.....	431
Configuring Validation Scenarios.....	433
Configuring Extensions.....	434
Configuring an Extensions Bundle.....	434
Customize Profiling Conditions.....	437
Preserve Style and Format on Copy and Paste from External Applications.....	437
Implementing an Author Extension State Listener.....	437
Implementing an Author Schema Aware Editing Handler.....	439
Configuring a Content Completion Handler.....	439
Configuring a Link target element finder.....	441
Configuring a custom Drag and Drop listener.....	444
Configuring a References Resolver.....	444
Configuring CSS Styles Filter.....	447
Configuring tables.....	447
Configuring an Unique Attributes Recognizer.....	455
Configuring an XML Node Renderer Customizer.....	455
Customizing the Default CSS of a Document Type.....	456
Document Type Sharing.....	457
Adding Custom Persistent Highlights.....	457
CSS Support in Author.....	457
Handling CSS Imports.....	458
Media Type <code>oxygen</code>	458
Standard W3C CSS Supported Features.....	458
Supported CSS Selectors.....	458

Supported CSS Properties.....	462
Supported CSS At-rules.....	468
Oxygen CSS Extensions.....	469
Additional CSS Selectors.....	469
Additional CSS Properties.....	471
Custom CSS Functions.....	475
Custom CSS Pseudo-classes.....	493
Builtin CSS Stylesheet.....	494
Example Files Listings - The Simple Documentation Framework Files.....	496
XML Schema files.....	496
sdf .xsd	496
abs .xsd	498
CSS Files.....	498
sdf .css	498
XML Files.....	499
sdf_sample.xml	499
XSL Files.....	501
sdf .xsl	501
Author Component.....	502
Licensing.....	502
Installation Requirements.....	503
Customization.....	503
Packing a fixed set of options.....	504
Deployment.....	504
Web Deployment.....	505
Sample SharePoint Integration of the Author Component.....	509
Author Component.....	509
Microsoft SharePoint®	510
Why Integrate the Author Component with SharePoint.....	510
Integration Adjustments.....	510
Getting Started.....	511
The Result.....	514
Frequently asked questions.....	514
Creating and Running Automated Tests.....	517

Chapter 9: API Frequently Asked Questions (API FAQ).....519

Difference Between a Document Type (Framework) and a Plugin Extension.....	521
Dynamically Modify the Content Inserted by the Writer.....	521
Split Paragraph on Enter (Instead of Showing Content Completion List).....	522
Impose Custom Options for Writers.....	523
Highlight Content.....	523
How Do I Add My Custom Actions to the Contextual Menu?.....	524
Adding Custom Callouts.....	525
Change the DOCTYPE of an Opened XML Document.....	528

Customize the Default Application Icons for Toolbars/Menus.....	528
Disable Context-Sensitive Menu Items for Custom Author Actions.....	528
Dynamic Open File in Oxygen XML Editor plugin Distributed via JavaWebStart.....	529
Change the Default Track Changes (Review) Author Name.....	530
Multiple Rendering Modes for the Same Author Document.....	530
Obtain a DOM Element from an AuthorNode or AuthorElement.....	530
Print Document Within the Author Component.....	531
Running XSLT or XQuery Transformations.....	531
Use Different Rendering Styles for Entity References, Comments or Processing Instructions.....	531
Insert an Element with all the Required Content.....	534
Obtain the Current Selected Element Using the Author API.....	534
Debugging a Plugin Using the Eclipse Workbench.....	535
Debugging an SDK Extension Using the Eclipse Workbench.....	535
Extending the Java Functionality of an Existing Framework (Document Type).....	536
Controlling XML Serialization in the Author Component.....	536
How can I add a custom Outline view for editing XML documents in the Text mode?.....	537
Dynamically Adding Form Controls Using a StylesFilter	540
Modifying the XML content on open.....	540

Chapter 10: Transforming Documents.....543

Output Formats.....	544
Transformation Scenario.....	545
Defining a New Transformation Scenario.....	545
XML transformation with XSLT.....	546
XML Transformation with XQuery.....	551
DITA OT Transformation.....	553
ANT Transformation	559
XSLT Transformation.....	561
XProc Transformation.....	563
XQuery Transformation.....	566
SQL Transformation.....	568
XSLT/XQuery Extensions.....	568
The Configure Transformation Scenario(s) Dialog.....	568
Duplicating a Transformation Scenario.....	570
Editing a Transformation Scenario.....	570
Batch Transformation.....	571
Built-in Transformation Scenarios.....	571
Transformation Scenarios View.....	571
Using the Oxygen WebHelp Plugin.....	574
Oxygen WebHelp Plugin for DITA.....	574
Integrating the Oxygen WebHelp Plugin with the DITA Open Toolkit.....	574
Registering the Oxygen WebHelp Plugin.....	574
Running a DITA Transformation Using the WebHelp Plugin.....	574
Database Configuration for DITA WebHelp with Feedback.....	575

Oxygen WebHelp Plugin for DocBook.....	576
Integrating the Oxygen WebHelp Plugin with the DocBook XSL Distribution.....	576
Registering the Oxygen WebHelp Plugin.....	576
Running a DocBook Transformation Using the WebHelp Plugin.....	576
Database Configuration for DocBook WebHelp with Feedback.....	577
XSLT Processors.....	577
Supported XSLT Processors.....	578
Configuring Custom XSLT Processors.....	579
Configuring the XSLT Processor Extensions Paths.....	579
XSL-FO Processors.....	580
The Built-in XSL-FO Processor.....	580
Add a Font to the Built-in FOP - The Simple Version.....	580
Add a Font to the Built-in FOP.....	581
Adding Libraries to the Built-in FOP.....	583
Chapter 11: Querying Documents.....	585
Running XPath Expressions.....	586
What is XPath.....	586
The XPath Dialog.....	586
The XPath/XQuery Builder View.....	589
Working with XQuery.....	591
What is XQuery.....	591
Syntax Highlight and Content Completion.....	591
XQuery Outline View.....	592
The XQuery Input View.....	593
XQuery Validation.....	594
Other XQuery Editing Actions.....	595
Transforming XML Documents Using XQuery.....	595
XQJ Transformers.....	596
Display Result in Sequence View.....	596
Advanced Saxon HE/PE/EE Transform Options.....	597
Updating XML Documents using XQuery.....	598
Chapter 12: Debugging XSLT Stylesheets and XQuery Documents.....	599
Overview.....	600
Layout.....	600
Control Toolbar.....	601
Information View.....	603
Context Node View.....	603
XPath Watch (XWatch) View.....	603
Breakpoints View.....	604
Messages View.....	605
Stack View.....	606

Output Mapping Stack View.....	607
Trace History View.....	607
Templates View.....	608
Nodes/Values Set View.....	609
Variables View.....	610
Multiple Output Documents in XSLT 2.0 and XSLT 3.0.....	611
Working with the XSLT / XQuery Debugger.....	611
Steps in a Typical Debug Process.....	612
Using Breakpoints.....	612
Inserting Breakpoints.....	612
Removing Breakpoints.....	613
Determining What XSLT / XQuery Expression Generated Particular Output.....	613
Debugging Java Extensions.....	614
Supported Processors for XSLT / XQuery Debugging.....	615

Chapter 13: Performance Profiling of XSLT Stylesheets and XQuery

Documents.....	617
Overview.....	618
Viewing Profiling Information.....	618
Invocation Tree View.....	618
Hotspots View.....	619
Working with XSLT/XQuery Profiler.....	619

Chapter 14: Working with Archives.....621

Browsing and Modifying Archive Structure.....	622
Working with EPUB.....	623
Create an EPUB.....	624
Publish to EPUB.....	624
Editing Files From Archives.....	624

Chapter 15: Working with Databases.....625

Relational Database Support.....	626
Configuring Database Data Sources.....	626
How to Configure an IBM DB2 Data Source.....	626
How to Configure a Microsoft SQL Server Data Source.....	627
How to Configure a Generic JDBC Data Source.....	628
How to Configure a MySQL Data Source.....	628
How to Configure an Oracle 11g Data Source.....	629
How to Configure a PostgreSQL 8.3 Data Source.....	630
Configuring Database Connections.....	632
How to Configure an IBM DB2 Connection.....	632
How to Configure a JDBC-ODBC Connection.....	633

How to Configure a Microsoft SQL Server Connection.....	633
How to Configure a MySQL Connection.....	634
How to Configure a Generic JDBC Connection.....	635
How to Configure an Oracle 11g Connection.....	635
How to Configure a PostgreSQL 8.3 Connection.....	636
Resource Management.....	637
Data Source Explorer View.....	637
Table Explorer View.....	640
SQL Execution Support.....	642
Drag and Drop from Data Source Explorer View.....	642
SQL Validation.....	644
Executing SQL Statements.....	644
Native XML Database (NXD) Support.....	644
Configuring Database Data Sources.....	645
How to Configure a Berkeley DB XML Data Source.....	645
How to Configure an eXist Data Source.....	645
How to Configure a MarkLogic Data Source.....	646
How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source.....	646
Configuring Database Connections.....	646
How to Configure a Berkeley DB XML Connection.....	646
How to Configure an eXist Connection.....	647
How to Configure a MarkLogic Connection.....	647
How to Configure an Documentum xDb (X-Hive/DB) 10 Connection.....	648
Data Source Explorer View.....	648
Oracle XML DB Browser.....	650
PostgreSQL Connection.....	650
Berkeley DB XML Connection.....	651
eXist Connection.....	654
MarkLogic Connection.....	655
Documentum xDb (X-Hive/DB) Connection.....	657
XQuery and Databases.....	660
Build Queries With Drag and Drop From Data Source Explorer View.....	660
XQuery Transformation.....	660
XQuery Database Debugging.....	662
Debugging with MarkLogic.....	662
Debugging with Berkeley DB XML.....	663
WebDAV Connection.....	663
How to Configure a WebDAV Connection.....	664
WebDAV Connection Actions.....	664
Actions Available at Connection Level.....	664
Actions Available at Folder Level.....	664
Actions Available at File Level.....	665
BaseX Support.....	665
Resource management.....	665
XQuery Execution.....	666

Chapter 16: Importing Data.....667

Introduction.....	668
Import from Database.....	668
Import Table Content as XML Document.....	668
Convert Table Structure to XML Schema.....	671
Import from MS Excel Files.....	671
Import from MS Excel 2007-2010 (.xlsx).....	673
Import from HTML Files.....	674
Import from Text Files.....	674

Chapter 17: Content Management System (CMS) Integration.....677

Integration with Documentum (CMS).....	678
Configure Connection to Documentum Server.....	678
How to Configure a Documentum (CMS) Data Source.....	678
How to Configure a Documentum (CMS) Connection.....	679
Known Issues.....	679
Documentum (CMS) Actions in the Data Source Explorer View.....	679
Actions Available on Connection.....	680
Actions Available on Cabinets / Folders.....	680
Actions Available on Resources.....	681
Transformations on DITA Content from Documentum (CMS).....	683
Integration with Microsoft SharePoint.....	683
How to Configure a SharePoint Connection.....	683
SharePoint Connection Actions.....	683
Actions Available at Connection Level.....	683
Actions Available at Folder Level.....	684
Actions Available at File Level.....	684

Chapter 18: Tools.....685

XML Digital Signatures.....	686
Overview.....	686
Canonicalizing Files.....	687
Certificates.....	688
Signing Files.....	688
Verifying the Signature.....	689

Chapter 19: Configuring Oxygen XML Editor plugin.....691

Configuring Options.....	692
Customized Default Options.....	692
Importing / Exporting Global Options.....	692
Preferences.....	693

Oxygen XML Editor plugin License.....	694
Fonts Preferences.....	694
Document Type Association Preferences.....	694
Locations Preferences.....	695
The Document Type Dialog.....	695
Document Type Sharing.....	702
Localizing Frameworks.....	703
Editor Preferences.....	705
Print Preferences.....	705
Edit modes Preferences.....	706
Format Preferences.....	714
Content Completion Preferences.....	717
Syntax Highlight Preferences.....	719
Open / Save Preferences.....	720
Templates Preferences.....	721
Spell Check Preferences.....	721
Document Checking Preferences.....	723
Mark Occurrences Preferences.....	723
Custom Validation Engines Preferences.....	724
CSS Validator Preferences.....	725
XML Preferences.....	725
XML Catalog Preferences.....	725
XML Parser Preferences.....	726
XML Instances Generator Preferences.....	728
XProc Engines Preferences.....	729
XSLT-FO-XQuery Preferences.....	730
Import Preferences.....	741
Data Sources Preferences.....	743
Data Sources Preferences.....	743
Download Links for Database Drivers.....	746
Table Filters Preferences.....	747
Archive Preferences.....	747
Custom Editor Variables Preferences.....	748
The Network Connection Settings Preferences.....	748
HTTP(S) Preferences.....	749
(S)FTP Preferences.....	749
Certificates Preferences.....	750
XML Structure Outline Preferences.....	751
Scenarios Management Preferences.....	751
Views Preferences.....	752
Reset Global Options.....	752
Scenarios Management.....	752
Editor Variables.....	753
Custom Editor Variables.....	755
Localization of the User Interface.....	755

Chapter 20: Upgrading Oxygen XML Editor plugin.....	757
Upgrading Oxygen XML Editor plugin on Windows / Linux.....	758
Upgrading Oxygen XML Editor plugin on Mac OS X.....	758
Chapter 21: Common Problems.....	759
Oxygen XML Editor plugin Takes Several Minutes to Start on Mac.....	760
XSLT Debugger Is Very Slow.....	760
Syntax Highlight Not Available in Eclipse Plugin.....	760
Problem Report Submitted on the Technical Support Form.....	760
Signature verification failed error on open or edit a resource from Documentum.....	761
Compatibility Issue Between Java and Certain Graphics Card Drivers.....	761
An Image Appears Stretched Out in the PDF Output.....	761
The DITA PDF Transformation Fails.....	762
Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x.....	763
JPEG CMYK Color Space Issues.....	763
MSXML 4.0 Transformation Issues.....	763
Glossary.....	765
Chapter 22: Using the WebApp Reviewer.....	767
The Review Mode.....	768
The Edit Mode.....	768

Chapter 1

Introduction

Topics:

- [Key Features and Benefits of Oxygen XML Editor plugin](#)

Welcome to the User Manual of Oxygen XML Editor plugin 15.2!

Oxygen XML Editor plugin 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 plugin 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 plugin offers support to generate output to multiple target formats, including: *PDF*, *PS*, *TXT*, *HTML*, *JavaHelp* and *XML*.

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 plugin

Multiplatform availability: Windows, Mac OS X, Linux, Solaris	Non blocking operations, you can perform validation and transformation operations in background
Visual WYSIWYG XML editing mode based on W3C CSS stylesheets.	Visual DITA Map editor
Closely integration of the DITA Open Toolkit for generating DITA output	Support for latest versions of document frameworks: DocBook and TEI.
Support for XML, XML Schema, 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 schemas, and NVDL schemas
Multiple built-in validation engines (Xerces, libxml, MSXML 4.0, MSXML.NET) and support for custom validation engines (Saxon SA, 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.
Visual schema editor with full and logical model views	Generate HTML documentation from XML Schemas
Ready to use FOP support to generate PDF or PS documents	XInclude support
Context sensitive content assistant driven by XML Schema, Relax NG, DTD, NVDL or by the edited document structure enhanced with schema annotation presenter	New XML document wizards to easily create documents specifying a schema or a DTD
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
Easy error tracking - locate the error source by clicking on it	Easy configuration for external FO Processors
Apply XSLT and FOP transformations	XPath search, evaluation and debugging 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
Batch validate selected files in project	Canonicalize and sign documents
Configurable actions key bindings	Associate extensions with editors provided by the Oxygen XML Editor plugin plugin.
XSLT Debugger with Backmapping support	XSLT Profiler
XQuery Debugger with Backmapping support	XQuery Profiler
Model View	Attributes View
XQuery 1.0 support	WSDL analysis and SOAP requests support
XSLT 2.0 / 3.0 full support	XPath 2.0 / 3.0 execution and debugging support

Document folding	Spell checking supporting English, German and French including locals
XSLT refactoring actions	Generate large sets of sample XML instances from XML Schema
Pretty-printing of XML files	Drag&drop support
Outline view in sync with a non well-formed document	

Chapter

2

Installation

Topics:

- [Installation Requirements](#)
- [Installation Instructions](#)
- [Making Oxygen XML Editor plugin Portable](#)
- [Obtaining and Registering a License Key](#)
- [Unregistering the License Key](#)
- [Upgrading Oxygen XML Editor plugin](#)
- [Checking for New Versions](#)
- [Uninstalling the Oxygen XML Editor plugin](#)
- [Performance Problems](#)

This section explains platform requirements and installation procedures. It also provides instructions on how to obtain and register a license key, how to perform upgrades and uninstall Oxygen XML Editor plugin.

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

Installation Requirements

This section presents the details about the platform and environment requirements necessary to install and run Oxygen XML Editor plugin.

Platform Requirements

The run-time requirements of Oxygen XML Editor plugin are the following:

- CPU (processor): minimum - Intel Pentium III™/AMD Athlon™ class processor, 500 *Mhz*; recommended - Dual Core class processor.
- Computer memory: minimum - 512 MB of RAM (1 GB on Windows Vista™, Windows 7, and Mac OS) ; recommended - 2 GB of RAM.
- Hard disk space: minimum - 300 MB free disk space ; recommended - 500 MB free disk space.

Operating System

The operating system requirements of Oxygen XML Editor plugin are the following:

Windows	Windows XP, Windows Vista, Windows 7, Windows 2003, Windows Server 2008, Windows Server 2012.
Mac OS	Mac OS X version 10.5 64-bit or later.
Unix/Linux	Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.6.0 or later from Oracle (formerly from Sun).

Environment Requirements

This section specifies the Java platform requirements and other tools that may be needed to install Oxygen XML Editor plugin.

Tools

Installation packages are supplied in compressed archives. Ensure you have installed a suitable archive extraction utility with which to extract the archive. The MD5 sum is available at [the Download page](#) for every archive. You should check the MD5 sum of the downloaded archive with an MD5 checking tool available on your platform.

Java Virtual Machine Prerequisites

Prior to installation, ensure that the latest stable Eclipse version (available at the release date of Oxygen XML Editor plugin) is installed on your computer. The current Eclipse version number is 3.7.

Oxygen XML Editor plugin supports only official and stable Java virtual machine versions 1.6.0 and later from Sun/Oracle (available at <http://www.oracle.com/technetwork/java/javase/downloads/index.html>) and from Apple Computer. The Java Virtual Machine from Apple is pre-installed on Mac OS X computers. For Mac OS X, Java Virtual Machine updates are available at the Apple website. Oxygen XML Editor plugin may work very well with JVM implementations from other vendors but the eventual incompatibilities will not be solved in further Oxygen XML Editor plugin releases. Oxygen XML Editor plugin *does not work with the GNU libgcj Java virtual machine*.

Installation Instructions

Before proceeding with the following instructions, please ensure that your system complies with the prerequisites detailed in the installation requirements.

The installation kits and the executable files packaged inside the installation kits were checked before publication with an antivirus program to make sure they are not infected with viruses, trojan horses or other malicious software.

Eclipse 3.4 - 4.2 Plugin Installation - The Update Site Method

Installation procedure for the Eclipse plugin in Eclipse 3.4 - 4.2 with the Update Site method.

1. Start Eclipse.
2. Choose the **Help > Software Updates > Available Software** menu option.
3. Press **Add Site** in the **Available Software** tab of the **Software Updates** dialog.
4. Enter `http://www.oxygenxml.com/InstData/Editor/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog.
5. Press **OK**.
6. Select the **oXygen XML Editor** checkbox.
7. Press **Install**.
8. Press the **Next** button in the following install pages.
9. You must accept the Eclipse restart confirmation.
10. When prompted for a license key, paste the license information received in the registration email.

This will happen when you use one of the wizards to create an XML project or document, when you open or create a document associated with Oxygen XML Editor plugin or when accessing the Oxygen XML Editor plugin Preferences.

The Oxygen XML Editor plugin is installed correctly if you can *create an XML project with the New Project wizard* of the plugin started from menu File -> New -> Other -> oXygen -> XML Project.

Eclipse 3.4 - 4.2 Plugin Installation - The Zip Archive Method


The steps for installing the Eclipse plugin in Eclipse 3.4 - 4.2 with the Zip Archive method.

1. [Download](#) the zip archive with the plugin.
2. Unzip the downloaded zip archive in the `dropins` subdirectory of the Eclipse install directory.
3. Restart Eclipse.

Eclipse should display an entry `com.oxygenxml.editor (15.2)` in the list available from Window - Preferences - Plug-in Development - Target Platform.

Making Oxygen XML Editor plugin Portable

To create a portable distribution of Oxygen XML Editor plugin that you can keep on an USB stick, follow these instruction:

- For Windows:
 - go to www.oxygenxml.com/InstData/Editor/Windows/VM/oxygen.zip and save the ZIP archive on your disk;
 -  **Note:** The ZIP includes Java Virtual Machine.
 - unpack the ZIP and copy the `license.txt` file to the unpacked directory;

- use `.exe` to start the application;
- For Mac OS X:
 - go to http://www.oxygenxml.com/download_oxygenxml_editor.html?os=MacOSX and download the MAC distribution;
 - 📄 **Note:** The MAC distribution does not include Java Virtual Machine.
 - unpack the ZIP and copy the `license.txt` file to the unpacked directory;
 - use `Oxygen.app` to start the application. Java will be installed automatically in case it is not installed already.
- For Linux and Ubuntu:
 - Download and install a Linux kit on one computer and copy the installation directory to the USB/network drive to distribute it the everyone:
 - go to http://www.oxygenxml.com/download_oxygenxml_editor.html?os=Linux and download the Linux kit that corresponds to your platform architecture (32 bit/ 64 bit);
 - 📄 **Note:** The ZIP includes Java Virtual Machine.
 - install it as a normal user (no `root` or `sudo`) in the user home and copy the `license.txt` file to the installation directory;
 - to start the application, use `sh`. Java is not needed on the system.
 - 📄 **Note:** The `.desktop` files will not work because the installation path is hard coded.
 - Download and unpack the all platforms archive(`tar.gz`), download Java for Linux and place it in the Oxygen directory:
 - go to http://www.oxygenxml.com/download_oxygenxml_editor.html?os=All and download the all platforms kit;
 - 📄 **Note:** Java Virtual Machine is not included.
 - go to <http://www.oracle.com/technetwork/java/javase/downloads/index.html>, download and unpack/install Linux JRE;
 - place the JRE in the Oxygen home directory, in a new directory named `jre`. The java executable should be `Oxygen\jre\bin\java`;
 - copy the `license.txt` file to the Oxygen directory;
 - use `sh oxygen.sh` to start the application.
 - 📄 **Note:** Java is not needed on the system. You have it bundled it with Oxygen XML Editor plugin.

Obtaining and Registering a License Key

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

For demonstration and evaluation purposes, a time limited license is available upon request at the [Oxygen XML Editor plugin website](#). 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 aspects. Thereafter, the software is disabled. To further use it, you must purchase a permanent license. For special circumstances, if a trial period greater than 30 days is required, please contact support@oxygenxml.com.

For definitions and legal details of the license types, consult the End User License Agreement received with the license key.



Note: It is also available at <http://www.oxygenxml.com/eula.html>, or in the **About** dialog box.

There are several ways to register a license, depending on its type and use case:

- using the **Register** dialog of Oxygen XML Editor plugin (the most common case) to register a named-user based or concurrent license;
- storing the license key into a file (either text or xml) that is copied to the installation directory of Oxygen XML Editor plugin.

At start-up, Oxygen XML Editor plugin looks for a valid license key in the following locations, in this order:

- *in the xml file registration;*
- *in the text file registration;*
- in its internal settings files created after you used the **Register** dialog to validate a *named-user based* or *concurrent license*.



Note: In case Oxygen XML Editor plugin is not licensed you are not able to edit documents or use any Oxygen XML Editor plugin specific actions. To enable the plugin, go to **Window > Preferences > <oXyegn/>**, click **Register** and enter your license key.

Named User License Registration

1. Save a backup copy of the message containing the new license key.
2. Start Oxygen XML Editor plugin.
3. Copy to the clipboard the license text as explained in the message.
4. If this is a new install of Oxygen XML Editor plugin, the registration dialog is displayed automatically at start-up. In the case you already used Oxygen XML Editor plugin and obtained a new license, go to **Window > Preferences > oXygen, Register** button.

The screenshot shows the 'Obtain a license key' dialog box. It contains three options for obtaining a license key, each with a corresponding button: 'Request a free 30-day trial license key' (Request a TRIAL license...), 'Purchase a permanent license key' (BUY Now...), and 'If you have a registration code, obtain a license key' (Request license for registration code...). Below these are two radio buttons: 'Use a license key' (selected) and 'Use a license server'. A bold instruction states: 'After you received the license key (either trial or permanent) paste it below. Note that the license key, usually received in a registration email, is composed of nine lines of text.' A 'Paste' button is located to the right of the text area. The text area contains the following text: 'Component=XML-Editor, XSLT-Debugger, Saxon-SA', 'Version=13', 'Number_of_Licenses=1', 'Date=12-08-2011', and 'Maintenance=1'. At the bottom are 'OK' and 'Cancel' buttons, and a help icon (?) is in the bottom left corner.

Figure 1: Registration Dialog

5. Select **Use a license key** as licensing method.
6. Paste the license text in the registration dialog.
7. Press the OK button.

Named User License Registration with Text File

1. Save the license key in a file named `licensekey.txt`.
2. Copy `licensekey.txt` in the installation folder of Oxygen XML Editor plugin or in the `lib` subfolder of the installation folder.
3. Start the Oxygen XML Editor plugin.

Named User License Registration with XML File

This procedure is designed to help system administrators register the application for multiple users, without the hassle of configuring the application licensing for each of them.

1. Depending on your license type, register Oxygen XML Editor plugin for the currently logged user, using one of the following procedures:
 - [Named User License Registration](#);
 - [Request a Floating License from a License Server Running as a Standalone Process](#) or [Request a Floating License from a License Server Running as a Java Servlet](#), if you have a floating license key.
2. Copy the `license.xml` file from the [preferences directory](#) of Oxygen XML Editor plugin to its installation directory (or its `lib` sub-directory).
3. For each installation of Oxygen XML Editor plugin (either on the same computer or on different computers) repeat step 2.

How Floating (Concurrent) Licenses Work

Floating licenses are *pooled* licenses that can be shared across a group of users. They are most often deployed when an organization has a group of users that only consume a license for a minority of their working hours. The licenses are returned back into the license pool as soon as they are released. Other users can then immediately reuse them.



Note: A user who runs two different distributions of Oxygen XML Editor plugin (for example Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single license.

The license management is done either by the Oxygen XML Editor plugin itself, or by the Oxygen license server:

- If you plan to use the application on machines running in the same local network, Oxygen XML Editor plugin can manage the licenses usage by itself. Different running instances of Oxygen XML Editor plugin communicate between them, using UDP broadcast on the 59153 port, to the 239.255.255.255 group. The registration procedure requires you to paste the license key in the license registration dialog. for further details, go to [Named User License Registration](#).
- If you plan to use the application on machines running in different network segments, use an Oxygen XML Editor plugin floating license server. A floating license server can be installed either as a [Java servlet](#) or as a [standalone process](#).

Setting up a Floating License Server Running as a Java Servlet

Setting up the floating license servlet.

Apache Tomcat 5.5 or higher is necessary. To get it, go to <http://tomcat.apache.org>.

1. Download the **Web ARchive (.war)** license servlet from one of the download URLs included in the registration email message.

2. Go to the Tomcat Web Application Manager page. In the **WAR file to deploy** section choose the WAR file and click the **Deploy** button. The *oXygen License Servlet* should be up and running, but there is no licensing information set.
3. To set the license key, log on the deployment machine, and go to the Tomcat installation folder (usually `/usr/local/tomcat`). Then go to the `webapps/oXygenLicenseServlet/WEB-INF/license/` folder and create a new file called `license.txt`. Copy the license text that was sent to you via e-mail into this file and save it.
4. It is recommended to password protect your pages using a Tomcat Realm. Please refer to the Tomcat Documentation for detailed info, like the [Realm Configuration HOW-TO - Memory Based Realm section](#).
5. Once you have defined a realm resource, you have to edit `webapps/oXygenLicenseServlet/WEB-INF/web.xml` file to configure user access rights on the license server. Note that Tomcat's standard security roles are used, i.e.: **standard** for licensing and **admin** or **manager** for the license usage report page.
6. By default, the license server is logging its activity in `/usr/local/tomcat/logs/oxygenLicenseServlet.log` file. To change the log file location, edit the `log4j.appender.R2.File` property from the `/usr/local/tomcat/webapps/oXygenLicenseServlet/WEB-INF/lib/log4j.properties` configuration file.
7. Restart *oXygen License Servlet* from the Tomcat Web Application Manager page.

Contact the Oxygen XML Editor plugin XML support staff at support@oxygenxml.com and ask for a new license key if:

- you have multiple license keys for the same Oxygen XML Editor plugin version and you want to have all of them managed by the same server;
- you have a multiple-user floating license and you want to split it between two or more license servers.

Report Page

You can access an activity report at

`http://hostName:port/oXygenLicenseServlet/license-servlet/report`.

It displays in real time the following 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 like:
 - 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:
 - licensed product;
 - registration name;
 - company name;
 - license category;
 - number of floating users;
 - Maintenance Pack validity.
- **Current license usage** - lists all currently acknowledged users:
 - user name;

- date and time when the license was granted;
- name and IP address of the computer where Oxygen XML Editor plugin runs;
- MAC address of the computer where Oxygen XML Editor plugin runs.



Note: The report is available also in XML format at `http://hostName:port/oxygenLicenseServlet/license-servlet/report-xml`.

Setting up a Floating License Server Running as a Standalone Process

Setting up the floating license server.



Important: Add `oxygenLicenseServer.exe` manually 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.

1. Download the license server installation kit for your platform from one of the download URLs included in the registration email message with your floating license key.
2. Unzip the install kit in a new folder.

The Windows installer *installs the license server as a Windows service*. It provides the following optional features that are not available in the other license server installers:

- Set the Windows Service name;
- Start the Windows service automatically at Windows startup;
- Create shortcuts on the Start menu for starting and stopping the Windows service manually.

If you use the zip archive on Windows you have to run the scripts provided in the archive for installing, starting, stopping and uninstalling the server as a Windows service.

The zip archive can be used for running the license server on any platform where a Java virtual machine can run (Windows, Mac OS X, Linux / Unix, etc).

3. Start the server using the startup script.

The startup script is called `licenseServer.bat` for Windows and `licenseServer.sh` for Mac OS X and Unix / Linux. It has 2 parameters:

- `licenseDir` - the path of the directory where the license files will be placed. Default value: `license`.
- `port` - the TCP port number used to communicate with Oxygen XML Editor plugin instances. Default value: **12346**.

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

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

The following is an example command line for starting the license server on Windows:

```
licenseServer.bat myLicenseDir 54321
```


The license folder must contain a text file called `license.txt` which must contain a single floating license key corresponding to the set of purchased floating licenses. If you have more than one floating license key for the same Oxygen XML Editor plugin version obtained from different purchases or you want to split a set of license keys between 2 different servers please contact us at support@oxygenxml.com to merge / split your license keys.

Install the License Server as a Windows Service

1. Download the Windows installer of the license server from the URL provided in the registration email message containing your floating license key.
2. Run the downloaded installer.
3. Enable the Windows service on the machine that hosts the license server, either during installation or at a later time with the service management batch scripts (`installWindowsService.bat`).


If you want to install, start, stop and uninstall manually the server as a Windows service you must run the following scripts from command line. On Windows Vista and Windows 7 you have to run the commands as Administrator.


- `installWindowsService.bat [serviceName]` - install 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]` - start the Windows service.
- `stopWindowsService.bat [serviceName]` - stop the Windows service.
- `uninstallWindowsService.bat [serviceName]` - uninstall the Windows service.

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

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

- `outLicenseServer.log` - server's standard output stream;
- `errLicenseServer.log` - server's standard error stream.

 **Note:** Before starting the server, the `JAVA_HOME` variable must point to the home folder of a Java runtime environment installed on your Windows system.

 **Note:** On Windows Vista and Windows 7 if you want to start or stop the Windows service with the Start menu shortcut called *Start Windows service / Stop Windows service* you have to run the shortcut as Administrator.

Common Problems

Here are the 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 has quit immediately after being started.

This problem usually happens because the license key has not been correctly deployed(license.txt file in the license folder). More details about this can found [here](#).

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

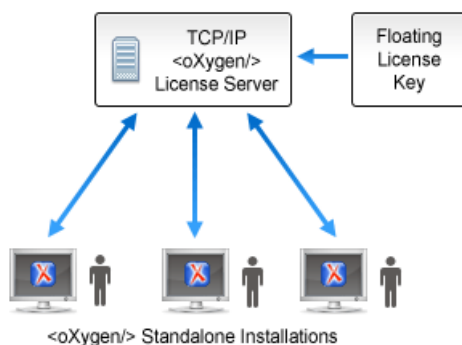


Figure 2: Floating License Server Running as a Standalone Process

1. Start the Eclipse platform.

2. Go to **Window > Preferences > oXygen > Register** .
The license dialog is displayed. Oxygen XML Editor plugin
3. Choose **Use a license server** as licensing method.
4. Select **Standalone server** as server type.
5. Fill-in the *Host* text field with the host name or IP address of the license server.
6. Fill-in the *Port* text field with the port number used to communicate with the license server.
7. Click the **OK** button.

If the maximum number of available licenses was not exceeded a license key is received from the floating license server and registered in Oxygen XML Editor plugin. To display the license details, go to the **About** dialog and select **Help**. If the maximum number of licenses was exceeded a warning dialog pops up letting you know about the problem. The message contains information about the users who requested and successfully received the floating licenses before exceeding the maximum number of licenses.

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

Starting with version 12.1, Oxygen XML Editor plugin can use a license server running as a Java Servlet to manage floating licenses.

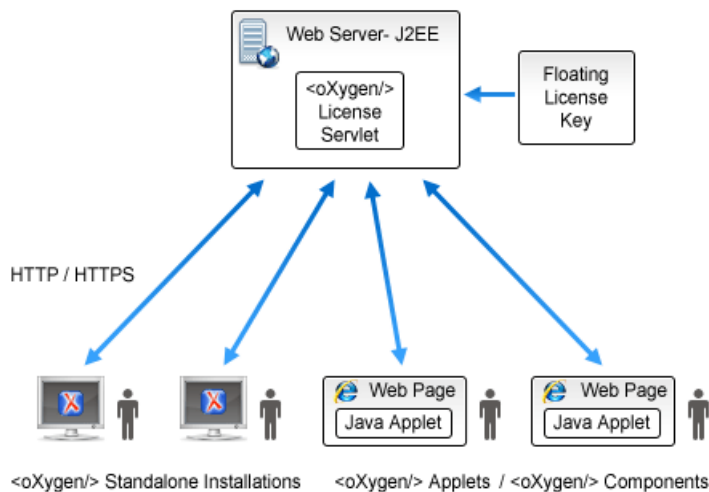


Figure 3: Floating License Server Running as a Servlet

1. Start the Eclipse platform.
2. Go to menu **Window > Preferences > oXygen > Register** .
The license dialog is displayed.
3. Choose **Use a license server** as licensing method.
4. Select **HTTP/HTTPS Server** as server type.
5. Fill-in the *URL* text field with the address of the license server.
The URL address has the following format:
`http://hostName:port/oXygenLicenseServlet/license-servlet`
6. Fill-in the *User* and *Password* text fields. Contact your server administrator to supply you this information.
7. Click the **OK** button.

If the maximum number of available licenses was not exceeded a license key is received from the floating license server and registered in Oxygen XML Editor plugin. The license details are displayed in the **About** dialog opened from the **Help** menu. If the maximum number of licenses was exceeded a warning dialog pops up letting you know about the problem. The message contains information about the users who requested and successfully received the floating licenses before exceeding the maximum number of licenses.



Note: Two different Oxygen XML Editor plugin instances (for example one standalone and one Eclipse plugin) run on the same machine, consume a single license key.

Release a Floating License

To manually release a floating license key to be returned to the server's pool of available license keys:

1. Go to the main Oxygen XML Editor plugin preferences panel .
2. Select **Register**.
3. Select **Use a license key** as licensing method.
4. Paste a Named User license key in the registration dialog. Leave the text area empty to return to the previously used license key, if any.
5. Press the **OK** button of the dialog.

License Registration with an Activation Code

If you have only an activation code and you want to register the associated license key you must request this license key by filling the registration code and other details associated with your license in a request form on the Oxygen XML Editor plugin website. The button **Request license for registration code** in the registration dialog available from menu **Window > Preferences > oXygen XML Editor > Register** opens this request form in the default Web browser on your computer.

Unregistering the License Key

Sometimes, you may need to unregister your license key. For example, to release a *floating license* to be used by other user and still use the current Oxygen XML Editor plugin instance with a Named User license, or to transfer your license key to other computer before other user starts using your current computer.

1. Go to menu **Windows > Preferences > oXygen > Register**
This displays the license registration dialog.
2. Make sure the text area for the license key is empty.
3. Make sure the option **Use a license server** is not selected.
4. Press the **OK** button of the dialog.
This displays a confirmation dialog.
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 of 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 plugin.

Upgrading Oxygen XML Editor plugin

From time to time, upgrade and patch versions of Oxygen XML Editor plugin are released to provide enhancements that rectify problems, improve functionality and the general efficiency of the application.

Any personal configuration settings and customizations are preserved by installing an upgrade or a patch.

Upgrading the Eclipse Plugin

1. Uninstall Oxygen XML Editor plugin (see [Uninstall procedure](#)).
2. Follow the [Installation instructions](#).
3. Restart the Eclipse platform.
4. Start the Oxygen XML Editor plugin to ensure that the application can start and that your license is recognized by the upgrade installation.
5. If you are upgrading to a major version, for example from 11.2 to 12.0, and you did not purchase a Maintenance Pack that covers the new major version (12.0) you will need to enter a new license for version 12 into the registration dialog that is shown when the plugin is started.
6. Go to menu **Window > Preferences > Plug-In Development > Target Platform**
7. The `com.oxygenxml.editor` list entry contains the version number of the installed plugin. If the previous version was 11.2.0 the list entry should now contain 12.0.0.

Checking for New Versions

Oxygen XML Editor plugin offers the option of checking for new versions at the <http://www.oxygenxml.com> site when the application is started.

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

You are informed automatically that a new version is available as follows:

- in case the version you are using is the same with the latest released version, you are notified when new builds are available at www.oxygenxml.com;
- in case the version you are using is the penultimate version (for example you are using version 14.1 and version 14.2 is available at www.oxygenxml.com), you are informed that a new version was released;
- in case at www.oxygenxml.com two versions are available, for example version 14.1 and version 15, and you are using version 14.1, you are informed that version 15 was released. Apart from this notification, if you do not have the latest build installed, you are also informed about new 14.1 builds;
- in case at www.oxygenxml.com both a minor and a major version is available, and you are using a version older than both of them, you are informed about the major version and also about the minor one.

Uninstalling the Oxygen XML Editor plugin

This section contains uninstallation procedures.

Uninstalling the Eclipse plugin



Caution:

The following procedure will remove Oxygen XML Editor plugin from your system. It will not remove the Eclipse platform. If you wish to uninstall Eclipse please see its uninstall instructions.

1. Choose the menu option **Help > About > Installation Details**.
2. Select Oxygen XML Editor plugin from the list of plugins.
3. Choose **Uninstall**.
4. Accept the Eclipse restart.

5. If you want to remove also the user preferences that were configured in the **Preferences** dialog you must remove the folder `%APPDATA%\com.oxygenxml` on Windows (usually `%APPDATA%` has the value `[user-home-dir]\Application Data`) / the subfolder `.com.oxygenxml` of the user home directory on Linux / the subfolder `Library/Preferences/com.oxygenxml` of the user home folder on Mac OS X.

Performance Problems

This section contains the solutions for some common problems that may appear when running Oxygen XML Editor plugin.

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. In case Oxygen XML Editor plugin throws an *Out Of Memory* error, go to **Window > Preferences > <oxygen/> > 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.

Chapter 3

Getting Started

Topics:

- [Getting Help](#)
- [Supported Types of Documents](#)
- [Perspectives](#)

This section gets you started with the editing perspectives of Oxygen XML Editor plugin.

Getting Help

To access the online help, which is available at any time while working in Oxygen XML Editor plugin, go to **Help > Help Contents > Oxygen plugin**

Supported Types of Documents

Oxygen XML Editor plugin provides a rich set of features for working with:

- XML documents and applications
- XSL stylesheets - transformations and debugging support
- Schema languages: XML Schema (versions 1.0 and 1.1), Relax NG (full and compact syntax), NVDL, Schematron, DTD
- Querying documents using XPath and XQuery
- Analyzing, composing and testing WSDL SOAP messages
- CSS documents

Perspectives

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

With Oxygen XML Editor plugin, 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.

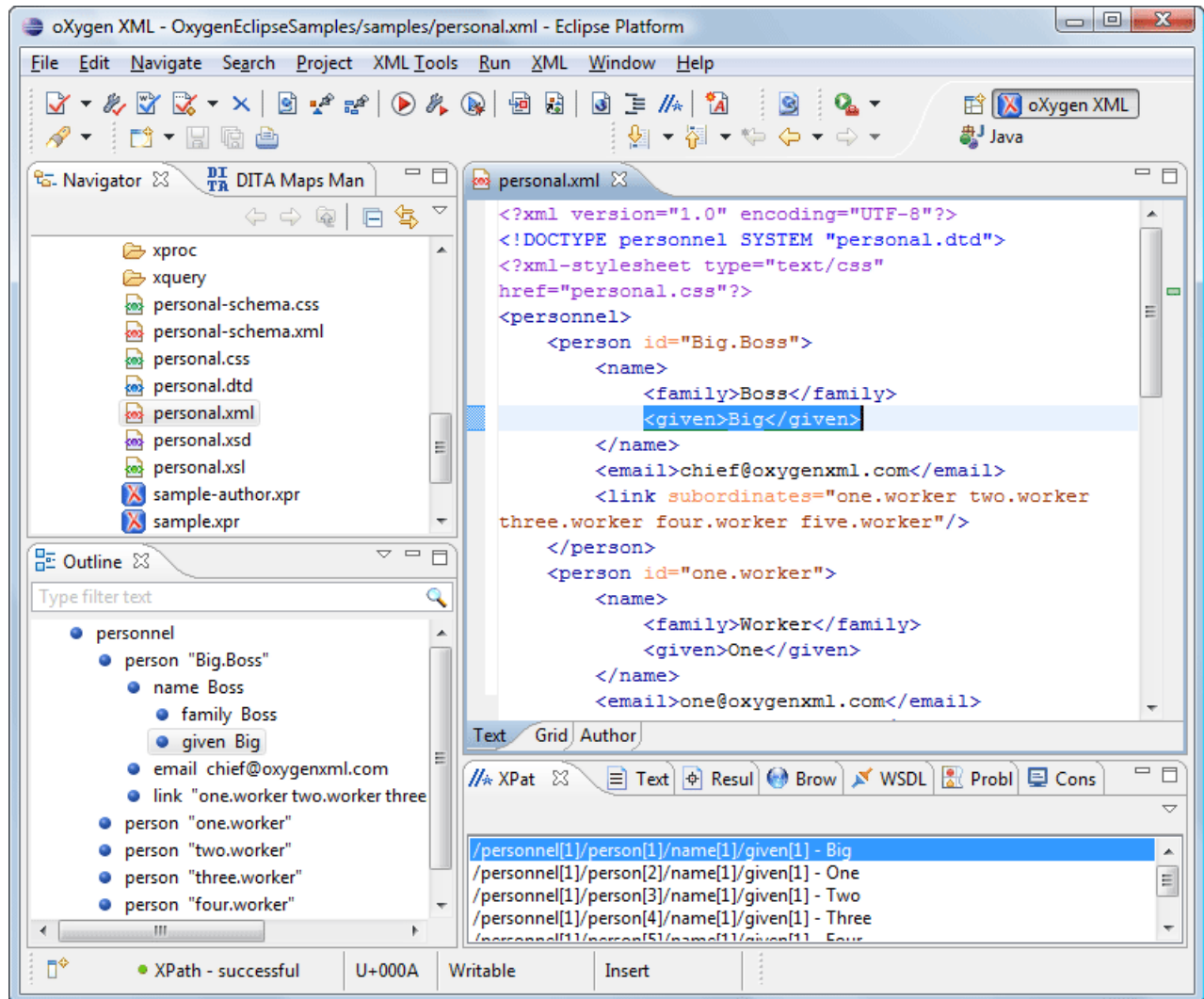
Database perspective

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.

Oxygen XML Editor plugin XML Perspective

To edit the content of your XML documents, use the Oxygen XML Editor plugin XML perspective.

Figure 4: Oxygen XML Editor plugin XML perspective



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.

The Oxygen XML Editor plugin Custom Menu

When the current editor window contains a document associated with Oxygen XML Editor plugin, a custom menu is added to the Eclipse menu bar. This custom menu is named after the document type: XML, XSL, XSD, RNG, RNC, Schematron, DTD, FO, WSDL, XQuery, HTML, CSS.

The Oxygen XML Editor plugin Toolbar Buttons

The toolbar buttons added by the Oxygen XML Editor plugin provide easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function.

The Editor Pane

The *editor pane*, or simply the *editor*, is where you edit your documents opened or created by the Oxygen XML Editor plugin Eclipse plugin. You know the document is associated with Oxygen XML Editor plugin from the special icon displayed in the editor's title bar which has the same graphic pattern painted with different colors for *different types of documents*.

This pane has three different modes of displaying and editing the content of a document available as different tabs at the bottom left margin of the editor panel: Text mode, Grid Mode, **Author** mode (CSS based tagless editor).

The Outline View

The **Outline** view displays a general tag overview of the currently edited XML Document. It also shows the correct hierarchical dependencies between elements. That makes it easier for you to be aware of the document structure and the way element tags are nested. It allows fast navigation of the document by displaying the start of the content of the child elements in the node of the parent element thus allowing to see quickly the content of an element without expanding it in the **Outline** tree. It also allows the user to insert or delete nodes using pop-up menu actions.

The outline view has the following functions: XML document overview, outliner filters, modification follow-up, document structure change, document tag selection.

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 wildcards (*, ?) and separate multiple patterns with commas.

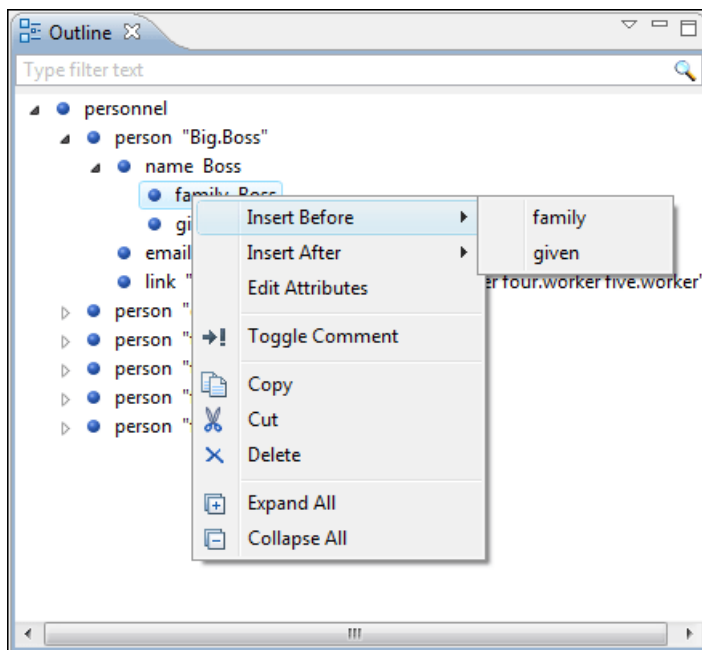


Figure 5: The Outline View

The Oxygen XML Editor plugin Text View

The Oxygen XML Editor plugin Text view is automatically showed in the views pane of the Eclipse window to display text output from XSLT transformations, FO processor's info, warning and error messages. It contains a tab for each file with text results displayed in the view.

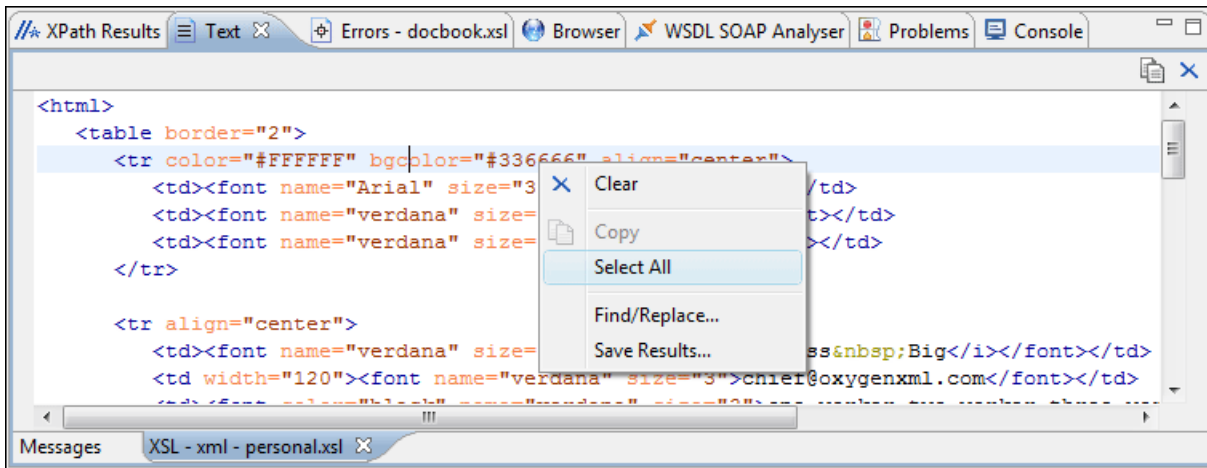


Figure 6: The Text View

The Oxygen XML Editor plugin Browser View

The Oxygen XML Editor plugin Browser view is automatically showed in the views pane of the Eclipse window to display HTML output from XSLT transformations. It contains a tab for each file with HTML results displayed in the view.

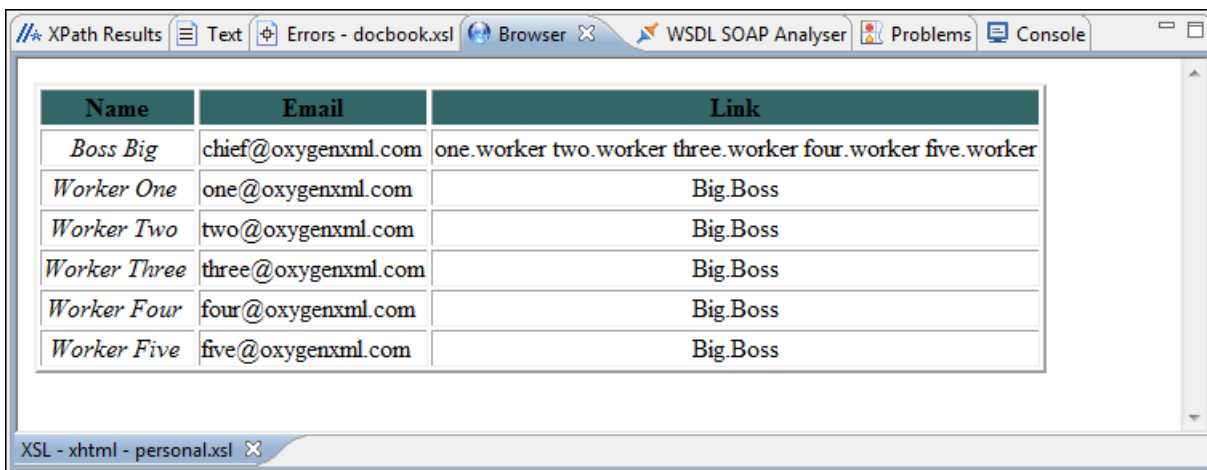


Figure 7: The Browser View

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*
-
-
- *Search References action*
- *SQL results*

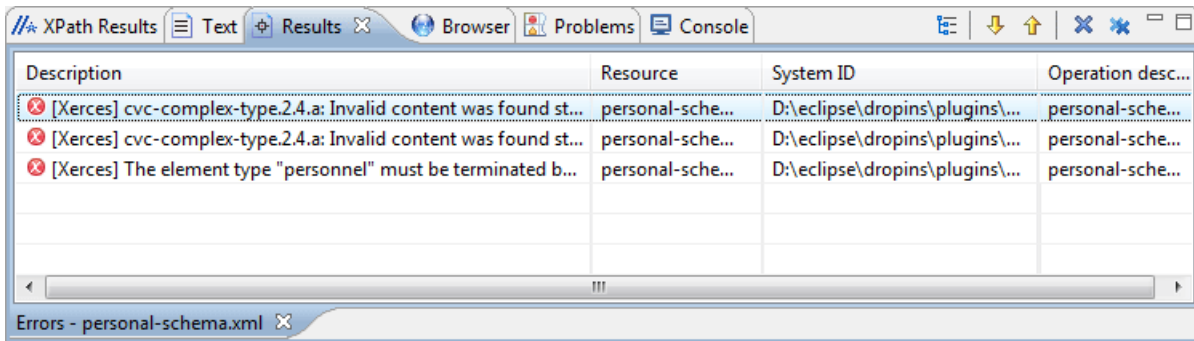


Figure 8: Results View

The view provides a toolbar with the following actions:

- remove actions: **Remove selected**, **Remove all** - these actions reduce the number of the messages from the view by removing them. It is useful when you do not want unimportant messages to clutter the view.

The actions available on the contextual menu are:

- **Remove selected** - Removes selected messages from the view.
- **Copy** - Copies the information associated with the selected messages:
 - the file path of the document that triggered the output message,
 - error severity (error, warning, info message an so on.),
 - name of validating processor,
 - the line and column in the file that triggered the message.
- **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*.
- **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.

The Oxygen XML Editor plugin XPath Results View



When you execute an XPath expression, Oxygen XML Editor plugin automatically displays the XPath Results view.
















Description	XPath location	Resource	System ID	Locati...
contr="false" id="Big.Boss"	/personnel[1]/person[1]	personal.xml	file:/D:/projects/...	5:5
contr="false" id="one.worker"	/personnel[1]/person[2]	personal.xml	file:/D:/projects/...	13:5
contr="false" id="two.worker"	/personnel[1]/person[3]	personal.xml	file:/D:/projects/...	21:5
contr="false" id="three.work...	/personnel[1]/person[4]	personal.xml	file:/D:/projects/...	29:5
contr="false" id="four.worker"	/personnel[1]/person[5]	personal.xml	file:/D:/projects/...	37:5
contr="false" id="five.worker"	/personnel[1]/person[6]	personal.xml	file:/D:/projects/...	45:5

Figure 9: The XPath Results View

Supported Editor Types

The Oxygen XML Editor plugin Eclipse plugin provides special Eclipse editors identified by the following icons:

-  - The XML documents icon;
-  - The XSL stylesheets icon;

-  - The XML Schema icon;
-  - The Document Type Definition schemas icon;
-  - The RELAX NG full syntax schemas icon;
-  - The RELAX NG compact syntax schemas icon;
-  - The Namespace-based Validation Dispatching Language schemas icon;
-  - The XSL:FO documents icon;
-  - The XQuery documents icon;
-  - The WSDL documents icon;
-  - The Schematron documents icon;
-  - The JavaScript documents icon;
-  - The Python documents icon;
-  - The CSS documents icon;
-  - The XProc documents icon;
-  - The SQL documents icon;
-  - The JSON documents icon.

XSLT Debugger Perspective

The XSLT Debugger perspective allows you to detect problems in an XSLT transformation process by executing the process step by step, in a controlled environment. The workspace is organized as an editing area supported 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 are displayed in *Text mode*. The (*Author mode* and *Grid mode*) are available only in *the Editor perspective*.

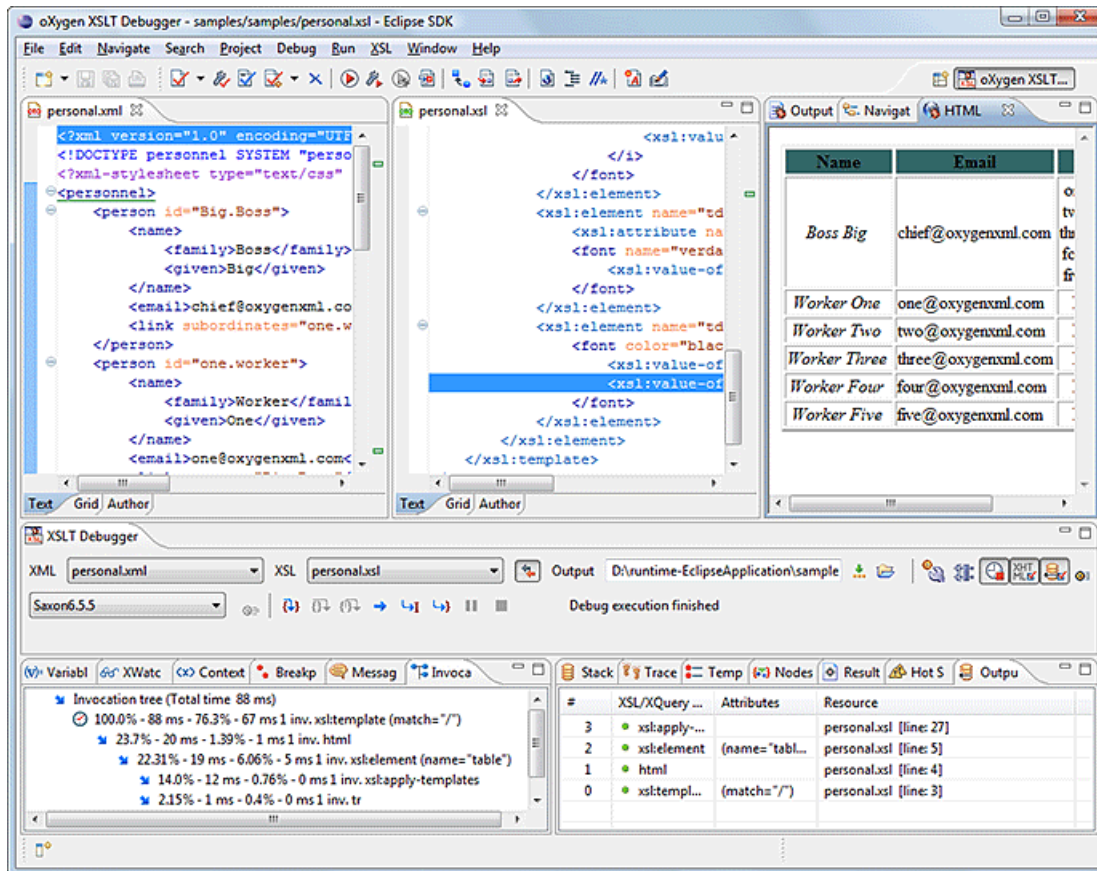



Figure 10: Oxygen XML Editor plugin - XSLT Debugger perspective

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

- 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 the developer 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 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.

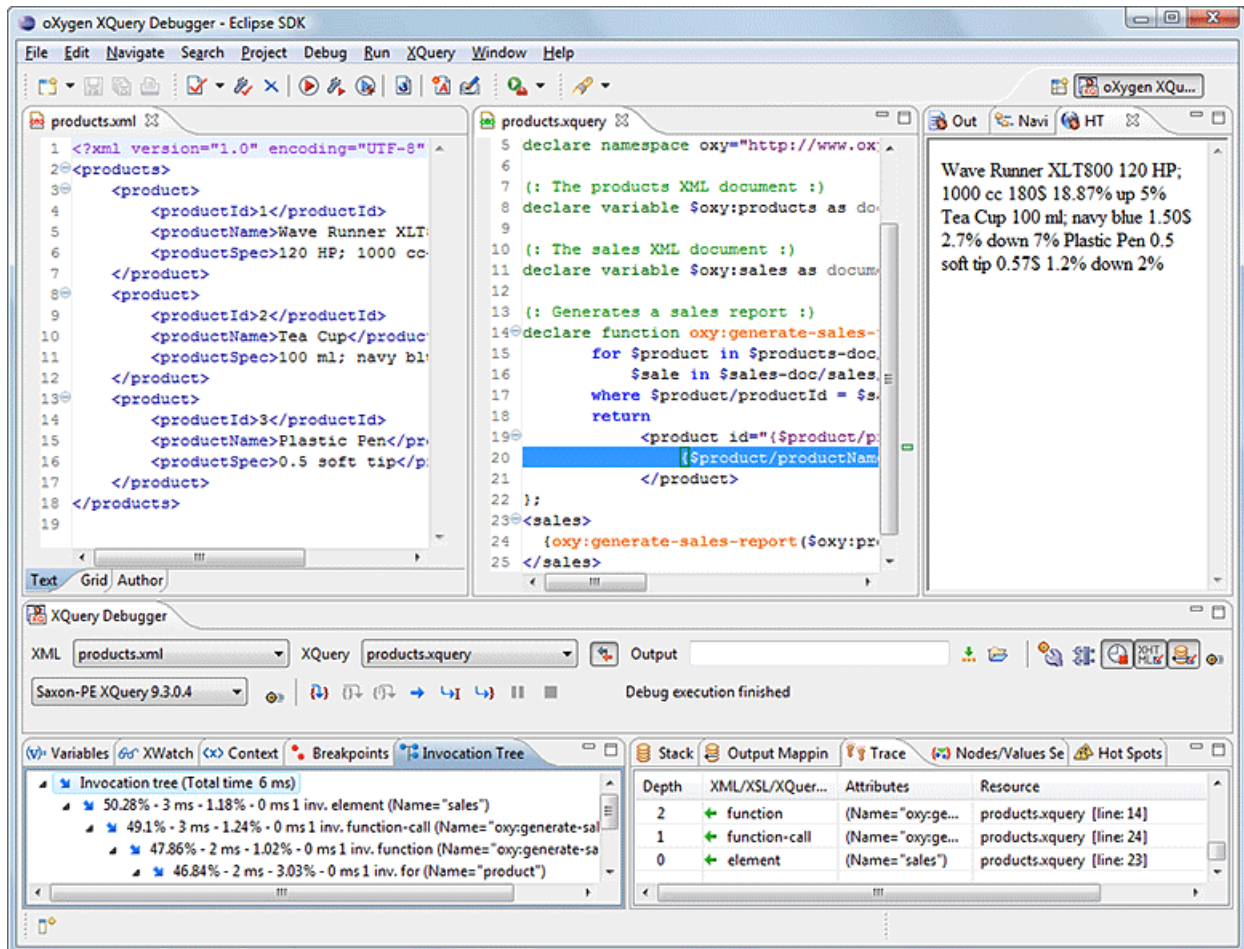


Figure 11: Oxygen XML Editor plugin XQuery Debugger perspective

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 plugin, go to http://www.oxygenxml.com/demo/XQuery_Debugger.html.

Oxygen XML Editor plugin Database Perspective

The **Database** perspective is similar to the **Editor** perspective. It 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 plugin. The non-XML capabilities of any database listed here are available also in the Academic and Professional editions of Oxygen XML Editor plugin 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 plugin.

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 plugin 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 plugin 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 plugin please *go to the Oxygen XML Editor plugin website*.



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

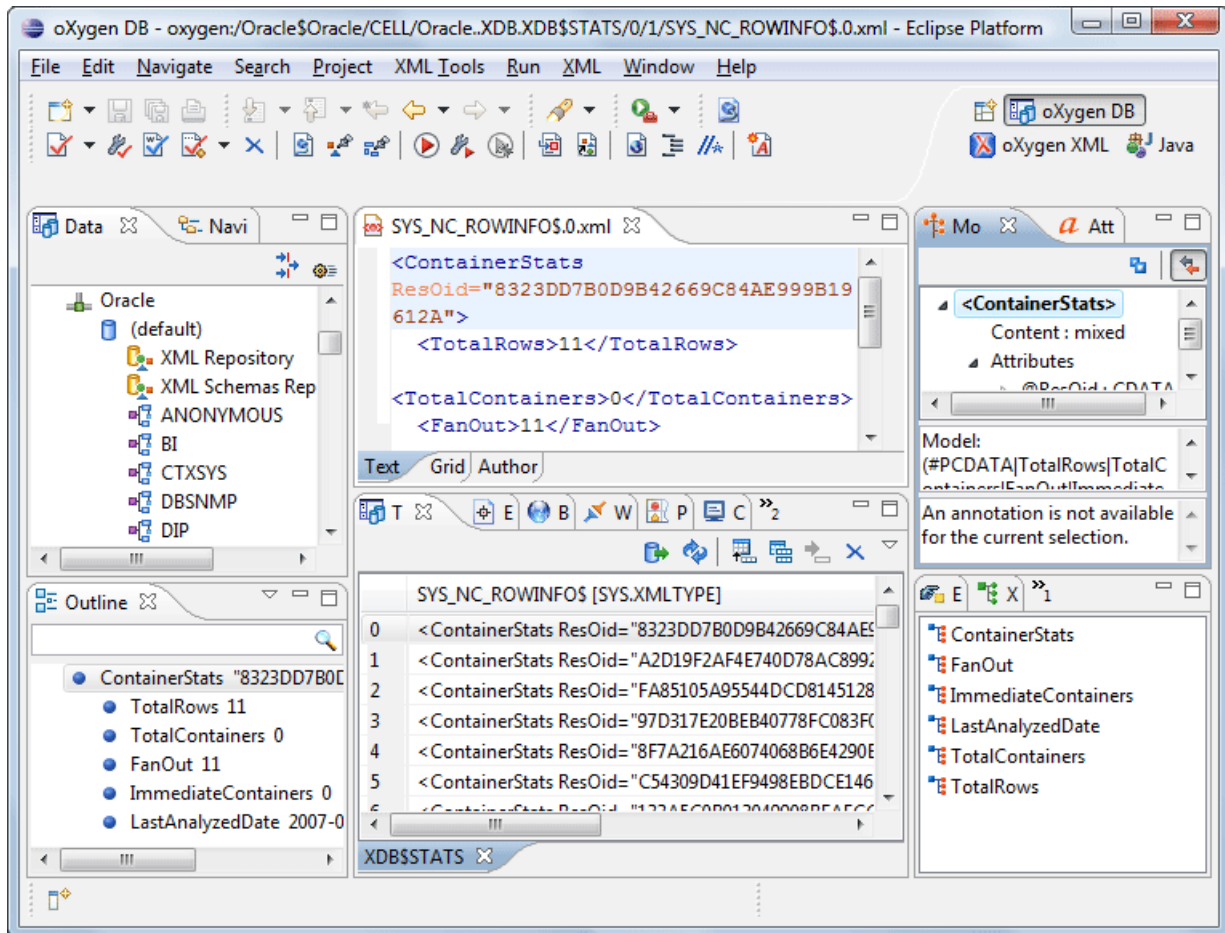


Figure 12: Database perspective

- Main menu - provides access to all the features and functions available within Oxygen XML Editor plugin.
- 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.

Chapter 4

Editing Modes

Topics:

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

To better suit any type of editing that you want to perform, Oxygen XML Editor plugin 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.

Text Editing Mode

The **Text** mode of Oxygen XML Editor plugin 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.

Finding and Replacing Text in the Current File

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

You can use a number of advanced views depending on what you need to find in the document you are editing and in even in your entire project. The [Find All Elements/Attributes dialog](#) searches through the structure of the current document for elements and attributes.

The Find All Elements / Attributes Dialog

To open the **Find All Elements / Attributes** dialog, go to **Edit > Find All Elements...** . It assists you in defining XML elements / attributes search operations on the current document.

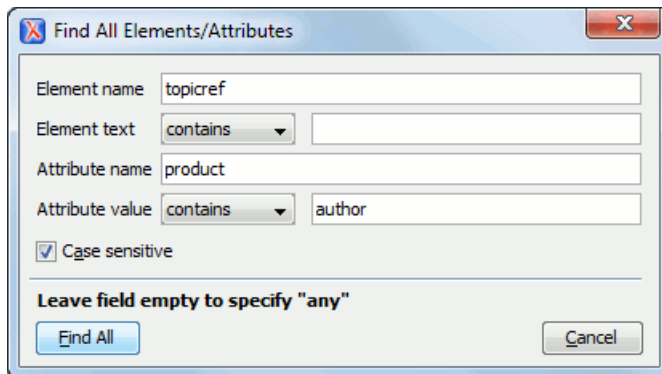


Figure 13: Find All Elements / Attributes dialog

The dialog can perform the following actions:

- Find all the elements with a specified name;
- Find all the elements which contain or not a specified string in their text content;
- Find all the elements which have a specified attribute;
- Find all the elements which have an attribute with or without a specified value.

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

The results of all the operations in the **Find All Elements / Attributes** dialog will be presented as a list in the message panel.

The following fields are available in the dialog:

- **Element name** - the target element name to search for. Only the elements with this exact name are returned. For any element name just leave the field empty;
- **Element text** - the target element text to search for. The combo box 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 **contains** in the combo box and leave the field empty. If you leave the field empty but select **equals** in the combo box, only elements with no text will be found. Select **not contains** to find all elements which do not have the specified text inside;
- **Attribute name** - the name of the attribute which needs to be present in the elements. Only the elements which have an attribute with this name will be returned. For any / no attribute name just leave the field empty;

- **Attribute value** - the combo box beside this field allows you to specify that you are looking for an exact or partial match of the attribute value. For any / no attribute value select **contains** in the combo box and leave the field empty. If you leave the field empty but select **equals** in the combo box, only elements that have at least an attribute with an empty value will be found. Select **not contains** to find all elements which have attributes without a specified value;
- **Case sensitive** - When this option is checked, operations are case sensitive.

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 (Meta on Mac OS) + "+"**, **Ctrl (Meta on Mac Os) + "-"**, **Ctrl (Meta on Mac OS) + 0** or **Ctrl (Meta on Mac OS) + Scroll Forward/Ctrl (Meta on Mac OS) + Scroll Backwards**.

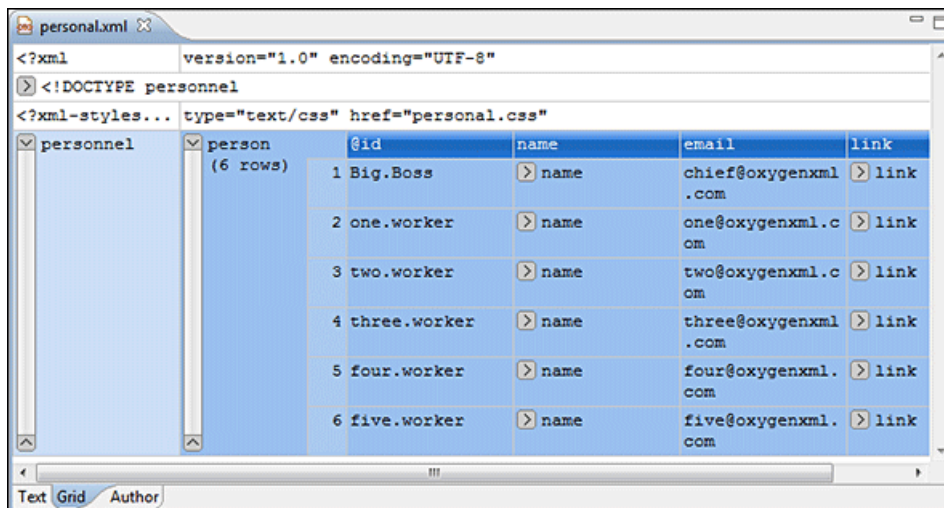


Figure 14: The Grid Editor

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. .

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 subtree 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 (Meta on Mac OS)- Space)** on your keyboard also displays the pop-up.

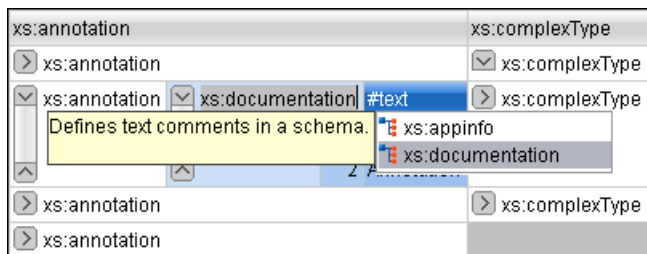


Figure 15: Content Completion in Grid Editor

To watch our video demonstration about some of the features available in the **Grid** editor, go to http://oxygenxml.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.

	test	table	tr	@id	first	last
			(3 rows)	1	10001	Jhon Doe
				2	10002	Mark Ewing
				3	10003	Dave Flint

Figure 16: Grid Layout

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

	test	table	tr	@id	first	last
				10001	Jhon	Doe
				10002	Mark	Ewing
				10003	Dave	Flint

Figure 17: Tree Layout

To switch between the two modes, go to **the contextual menu > 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 a dialog. The same set of key shortcuts used to select dialog components is also available in the **Grid** mode:

Table 1: Shortcuts in the Grid Mode

Key	Action
Tab	Moves the caret to the next editable value in a table row.
Shift + Tab	Moves the caret to the previous editable value in a table row.
Enter	Begins editing and lets you insert a new value. Also commits the changes after you finish editing.
Up Arrow/Page Up	Navigates toward the beginning of the document.
Down Arrow/Page Down	Navigates toward the end of the document.



Key	Action
Shift	Used with the navigation keys, creates a selection area. To extend this area with other nodes from a different part of the document, use the caret and the Ctrl (Meta on Mac OS) key.

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

- **Ctrl (Meta on Mac OS) + Up Arrow** - scrolls the grid upwards;
- **Ctrl (Meta on Mac OS) + Down Arrow** - scrolls the grid downwards;
- **Ctrl (Meta on Mac OS) + Left Arrow** scrolls the grid to the left;
- **Ctrl (Meta on Mac OS) + Right Arrow** 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 + Plus** and **NumPad + Minus** keys, or from the **Expand/Collapse** submenu of the contextual menu.



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 the table by a specific column. The sorting can be either ascending or descending. The icons for this pair of actions are  and .

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 analyses automatically the content and decides what type of sorting to apply. When a mixed set of values is present in the sorted column, a dialog is displayed allowing 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** row operation, or by selecting  **Insert row** from the **Table** contextual menu.

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 one, using the  **Insert column** action from the **Table** contextual menu.

Clearing the Content of a Column

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

Adding Nodes

Using the contextual menu you can add nodes before, after, or as last child of the currently selected node.


The sub-menus containing detailed actions are:

- **Insert before;**
- **Insert after;**
- **Append child.**

Duplicating Nodes

A quicker way of creating new nodes is to duplicate the existing ones. The action is available in the **Duplicate** contextual menu and in the **Document > Grid Edit > Duplicate** menu.

Refresh Layout

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

Start Editing a Cell Value

You can simply press **(Enter)** after you have selected the grid cell.

Stop Editing a Cell Value

You stop editing a cell value when you press **(Enter)**.

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 arrange different sections in your XML document in the **Grid** mode using drag and drop actions.

With drag and drop you can:

- 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 selection. To deselect one of the selected fragments, use **Ctrl (Meta on Mac OS) + Click**.

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 application 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 (Meta on Mac OS) + Click**. 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 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.

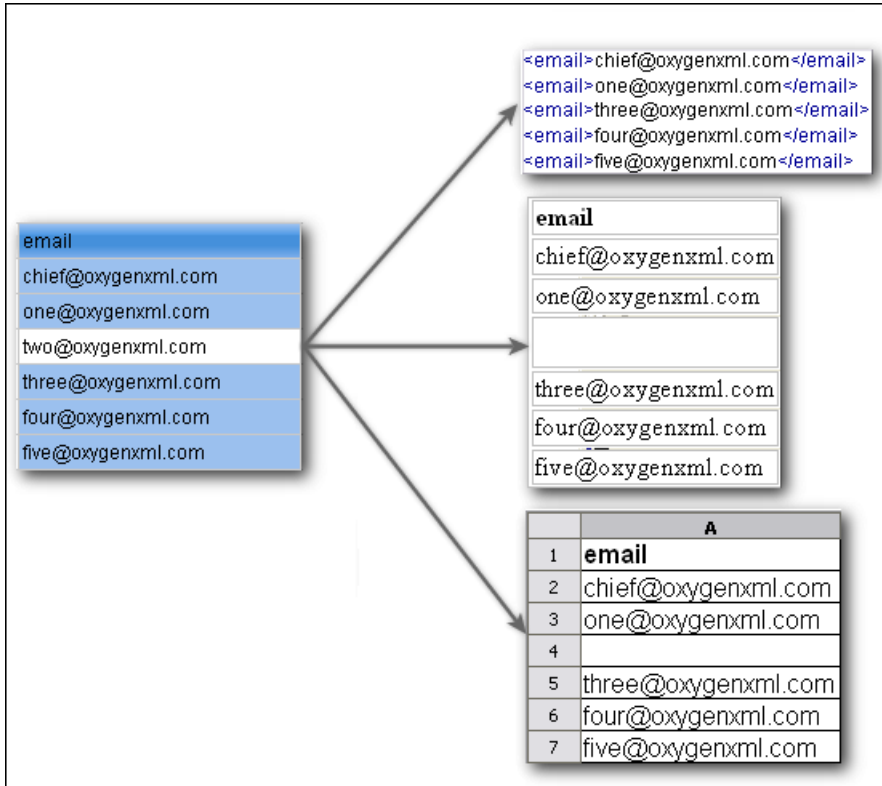


Figure 18: Copying from Grid to Other Editors

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.

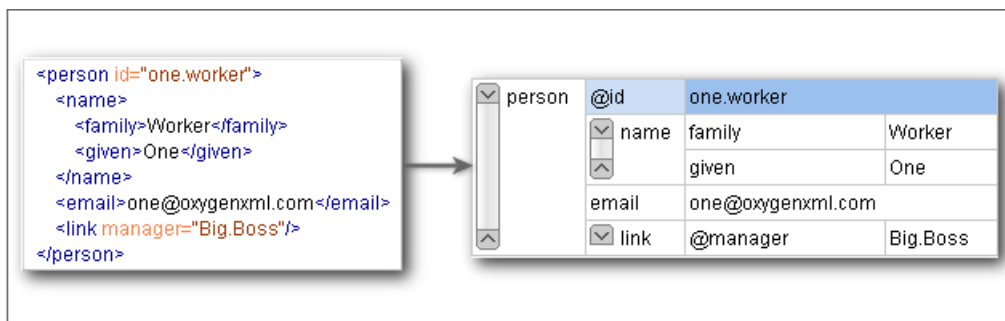


Figure 19: Copying XML Data into Grid

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.

Id	Email
Id1	Email1
Id2	Email2
Id3	Email3

@id	email
1 Big.Boss	chief@oxygenxml.com
2 Id1	Email1
3 Id2	Email2
4 Id3	Email3

Figure 20: 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 (Meta on Mac OS) + Shift + O** 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	#text
(9 rows)	1 عندما يريد العالم أن يتكلم، فهو يتحدث بلغة يونيكود.
	2 Quan el món vol conversar, parla Unicode
	3 כאשר 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 향한 대화, 유니코드로 하십시오
	8 Når verden vil snakke, snakker den Unicode
	9 Når verda ønskjer å snakke, talar ho Unicode

Figure 21: Default left to right text orientation

<?xml	"version="1.0" encoding="UTF-8"
sample	#text
(9 rows)	1 عندما يريد العالم أن يتكلم، فهو يتحدث بلغة يونيكود.
	2 Quan el món vol conversar, parla Unicode
	3 כאשר 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 향한 대화, 유니코드로 하십시오
	8 Når verden vil snakke, snakker den Unicode
	9 Når verda ønskjer å snakke, talar ho Unicode

Figure 22: Right to left text orientation

Author Editing Mode

This chapter presents the WYSIWYG like editor, called **Author** editor, 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 (what-you-see-is-what-you-get) editor in which the XML markup is not visible.

This tagless editor is available as the **Author** mode of the XML editor. To enter the **Author** 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.

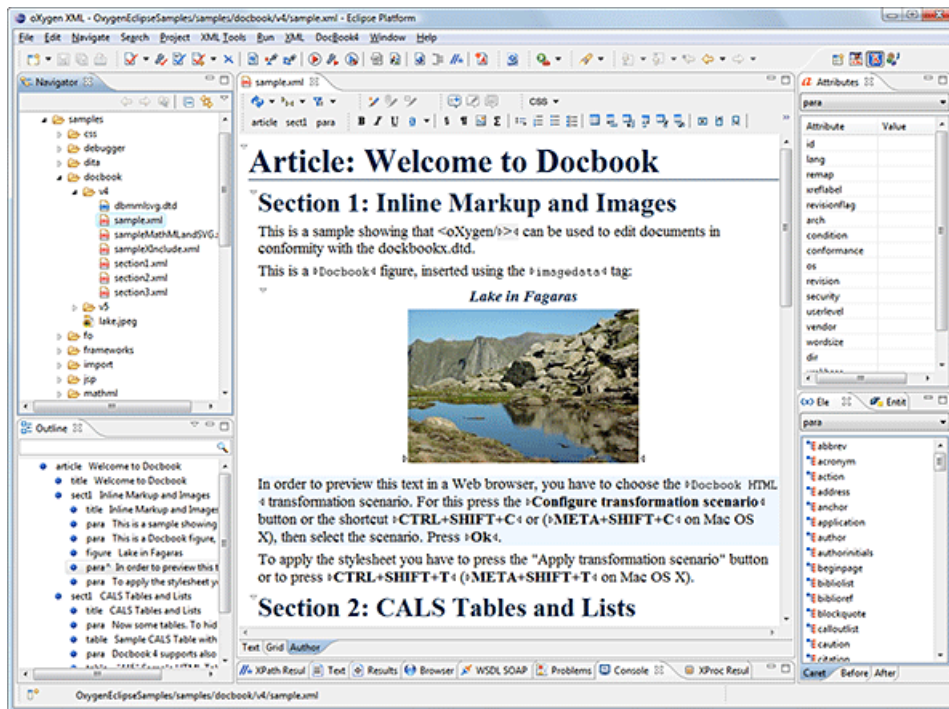


Figure 23: Author Editing Mode

Associating a CSS 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 like namespaces and custom extensions of the CSS specification are supported also.

The CSS specification is convenient for driving the tagless rendering of XML documents as it is an open standard maintained by the W3C consortium. A stylesheet conforming to this specification is easy to [develop and edit](#) in Oxygen XML Editor plugin as it is a plain text file with a simple syntax.

The association of such a stylesheet with an XML document is straightforward: an `xml-stylesheet XML` processing instruction with the attribute `type="text/css"` must be inserted at the beginning of the XML document. In case you do not want to alter your XML documents, you should set-up a *document type*.

For XHTML documents, there is a second method for the association of a CSS stylesheet: an element `link` with the `href` and `type` attributes in the head child element of the `html` element as specified in the [W3C CSS specification](#).

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 to the content authors. A *content author* does not need to have advanced knowledge about XML tags or operations like validation of XML documents or applying an XPath expression to an XML document. The author just plugs the framework set-up by the developer into the application and starts editing the content of XML documents without editing the XML tags directly.

The framework set-up by the developer is called *document type* and defines a type of XML documents by specifying all the details needed for editing the content of XML documents in tagless mode:


- the CSS stylesheet which 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 largely used today like DocBook, DITA, TEI, XHTML.

To watch our video demonstration about the basic functionality of the **Author** mode, go to http://oxygenxml.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-install-folder]/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 `xml-stylesheet` processing instructions. You can insert the processing instruction by editing the document or by using the  **Associate XSLT/CSS stylesheet** action.

The syntax of such a processing instruction is:

```
<?xml-stylesheet 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 referred 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.

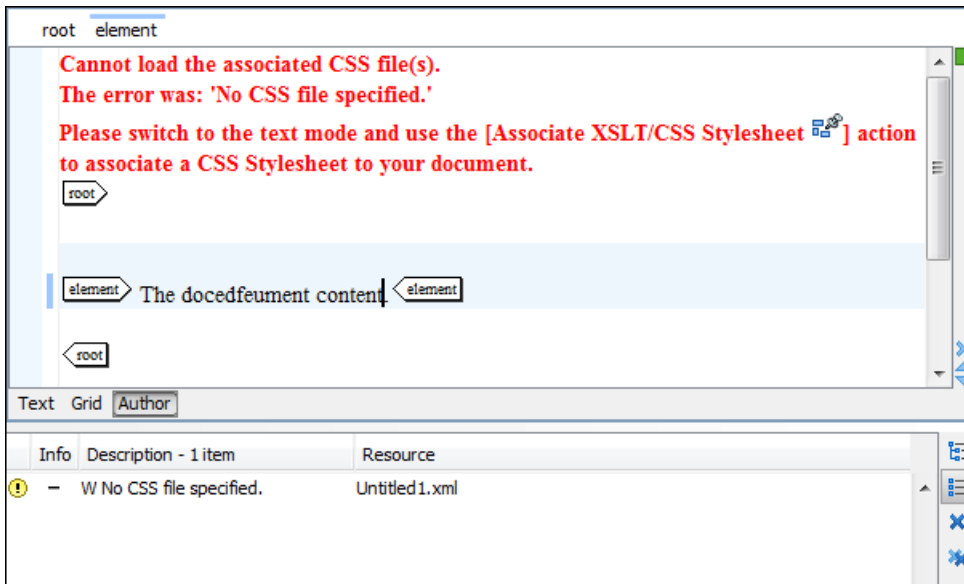


Figure 24: Document with no CSS association default rendering

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.*

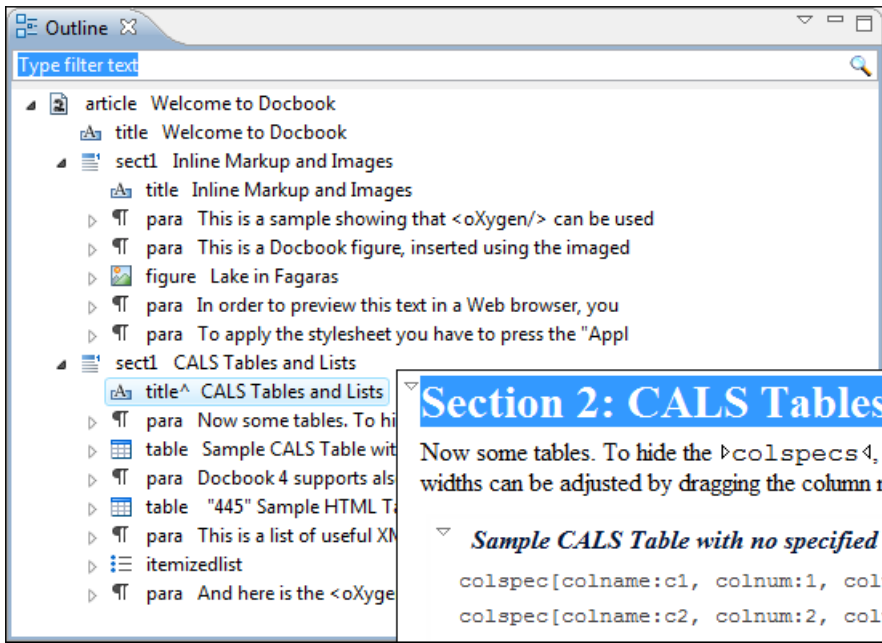


Figure 25: The Outline View

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 (Meta on Mac OS) + Click**.

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 (Meta on Mac OS))** 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 **View menu** on the Outline view's action bar:


-  **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** - A dialog is presented allowing the user to see and edit the attributes of the selected node.
- The **Append child**, **Insert before** and **Insert after** submenus allow 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.
- The **Cut**, **Copy** and **Delete** actions execute the same actions as the **Edit** menu items with the same name on the elements currently selected in the **Outline** tree (**Cut**, **Copy**, **Paste**).
- You can insert a well-formed element before, after or as a child of the currently selected element by accessing the **Paste before**, **Paste after** or **Paste as Child** actions.
- The **Toggle Comment** item encloses the currently selected element of the **Outline** tree in an XML comment, if the element is not commented, or removes the comment if it is commented.
- Using the **Rename Element** action 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.
- The **Expand All / Collapse All** actions 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.

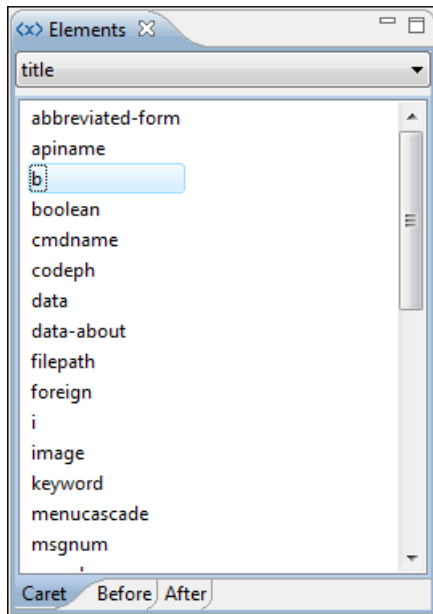


Figure 26: The Elements View


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.

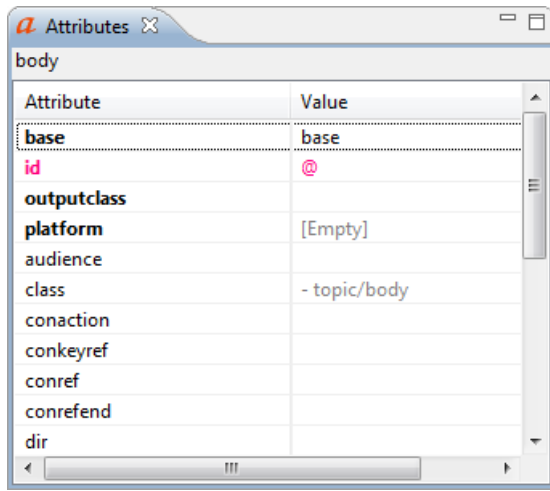


Figure 27: The Attributes View

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, in case the attribute does not appear in the edited document;
 - empty, in case 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:
 - in case 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;
 - in case 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;
 - in case the clipboard only contains text, the value of the selected attribute is modified.

To edit in place the attributes of an element in the editor panel, select the attribute and press **Alt + Enter** on your keyboard. This shortcut pops up a small window with the same content of 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.

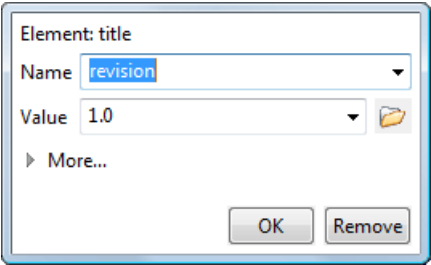


Figure 28: Edit attributes in place

The small right arrow button expands the list of possible attributes allowed by the schema of the document as in the **Attributes** view.

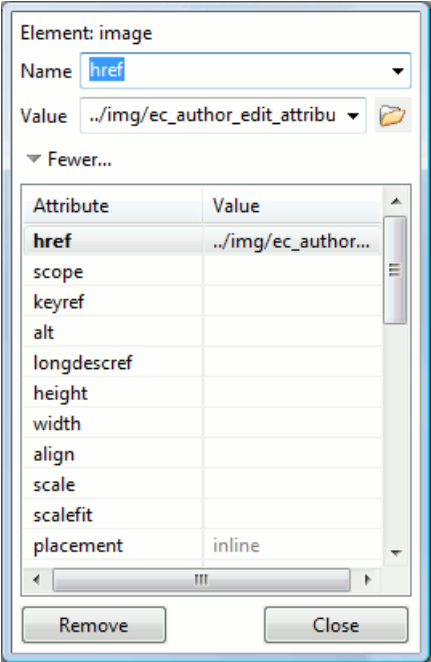


Figure 29: Edit attributes in place - full version

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.

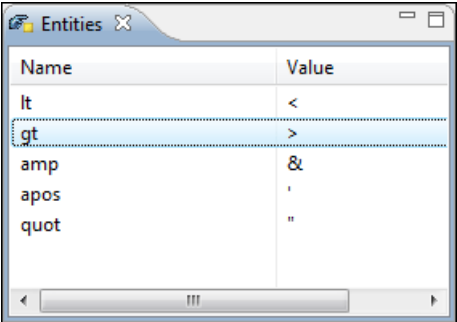


Figure 30: The Entities View

The Author Editor

This section explains the features of the CSS-driven WYSIWYG-like editor for XML documents.

Navigating the Document Content

Using the Keyboard

Oxygen XML Editor plugin 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 your 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 (Meta on Mac OS) + Right Arrow**, and **Ctrl (Meta on Mac OS) + Left Arrow**, respectively. Entities and hidden elements are skipped.

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 (Meta on Mac OS) + Alt + [**) and **Forward** (**Ctrl (Meta on Mac OS) + Alt +]**) buttons from the **Navigation** toolbar. The **Last Modification** (**Ctrl (Meta on Mac OS) + Alt + G**) button automatically takes you to the latest edited text.

Using the Breadcrumb Helpers

A left-hand side stripe paints a vertical thin light blue bar indicating the span of the element found at caret position. Also a top stripe called *breadcrumb* indicates the path from document root to the current element.

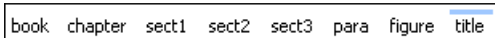


Figure 31: The breadcrumb in Editor view

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.

The tag names displayed in the breadcrumb can be customized with an Author extension class that implements `AuthorBreadcrumbCustomizer`. See the *Author SDK* for details about using it.

The **Append child**, **Insert before** and **Insert after** submenus of the top stripe 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 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.






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 (Cut, Copy, Paste, Delete). 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.

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 (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.







Using the Linking Support

When working on a suite of documents that refer to 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 plugin 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 a followable link to open the referred resource in an editor. The same effect can be obtained by using the action **Open file at caret** when the caret is in a followable link element.

To position the cursor at the beginning or at the end of the document you can use **Ctrl (Meta on Mac OS) + Home**, and **Ctrl (Meta on Mac OS) + End** respectively.

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. Later the cursor can go quickly to the marked position with a keyboard shortcut or with a menu item. This is useful to ease the navigation in a large document or to work on more than one document when the cursor must move between several marked positions.

A bookmark can be placed with:

- one of the menu items available on the menu **Edit > Bookmarks > Create**;

- the menu item **Edit > Bookmarks > Bookmarks Quick Creation (F9)**;
- the keyboard shortcuts associated with these menu items and visible on the menu **Edit > Bookmarks**.

A bookmark can be removed when a new bookmark is placed in the same position as an old one or with the action **Edit > Bookmarks > Remove All**. The cursor can go to a bookmark with one of the actions available on the menu **Edit > Bookmarks > Go to**.

Position Information Tooltip

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.

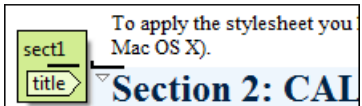


Figure 32: Before first block

- The caret is positioned between two block elements.

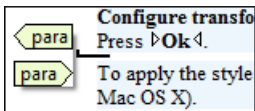


Figure 33: Between two block elements

- The caret is positioned after the last block element child of the current node.

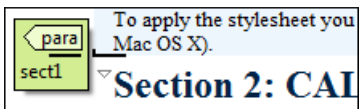


Figure 34: After last block

- The caret is positioned inside a node.

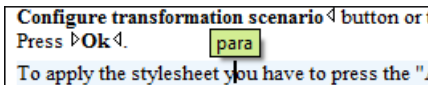


Figure 35: Inside a node

- The caret is positioned inside an element, before an inline child element.

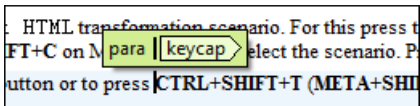


Figure 36: Before an inline element

- The caret is positioned between two inline elements.

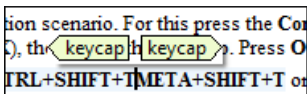


Figure 37: Between two inline elements

- The caret is positioned inside an element, after an inline child element.

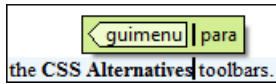



Figure 38: After an inline element

The nodes in the previous cases are displayed in the tooltip window using their names.


You can deactivate this feature by unchecking the **Options > Preferences > Editor / Author > Show caret position tooltip** check box. 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.

Displaying Referred Content

The references to entities, XInclude, and DITA conrefs are expanded by default in Author mode and the referred content is displayed. You can control this behavior from the [Author preferences page](#). The referred resources are loaded and displayed inside the element or entity that refers them, however the displayed content cannot be modified directly.

When the referred 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 referred content, you must open the referred resource in an editor. The referred resource can be opened quickly by clicking the link (marked with the icon ) which is displayed before the referred 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 referred resource is resolved through the XML Catalog set in **Preferences**.

The referred content is refreshed:

- automatically, when it is modified and saved from Oxygen XML Editor plugin;
- on demand, by using the [Refresh references action](#). Useful when the referred content is modified outside the Oxygen XML Editor plugin scope.

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).

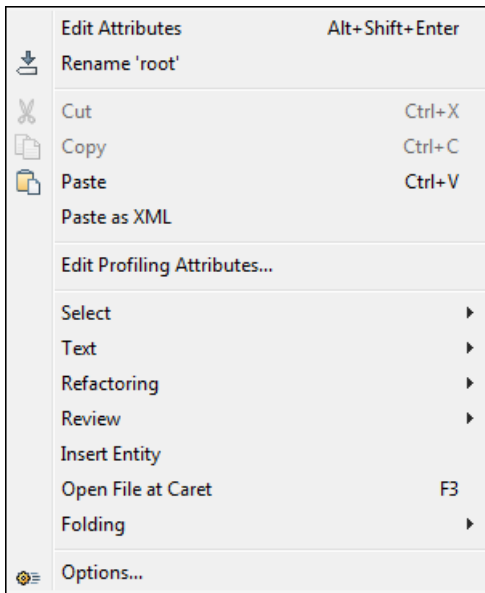


Figure 39: Contextual menu

The generic actions are:

- **Edit Attributes** - A pop-up window is displayed allowing you to manage the element attributes;
- **Rename** - The element from the caret position can be renamed quickly using the content completion window. If the *Allow only insertion of valid elements and attributes* schema aware option is enabled only the proposals from the content completion list are allowed, otherwise a custom element name can also be provided;
- **Cut, Copy, Paste** - Common edit actions with the same functionality as those found in the text editor;
- **Paste As XML** - Similar to **Paste** operation, except that the clipboard's content is considered to be XML;
- **Edit Profiling Attributes** - Allows you to select the profiling attributes;
- **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 selection content to lower case characters;
 - **Text > To Upper Case** - Converts the selection 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 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;
- **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;
- **Manage IDs** - Provides access to [searching and refactory actions for ID/IDREFS](#);
- **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
- **Open File at Caret** - Opens in a new editor panel the file with the path under 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 that is displayed contains a **Create new file** action which displays the **New** file dialog. This allows you to choose the type or the template for the file. If the action succeeds, the file is created with the referred 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 cursor position;
- **Options** - Opens the [Author options page](#).

Document type actions are specific to some document type. Examples of such actions can be found in the [Predefined document types](#) section.

Editing XML Documents in Author

This section details how to edit the text content and the markup of XML documents in **Author** mode. It explains also how to edit tables and MathML content 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 (Meta on Mac OS) + Space** 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, if any;
- 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.

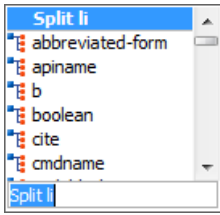


Figure 40: Content completion window

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 with the action  **Remove All Markup** available on the submenu **Refactoring** of the contextual menu and on the toolbar **XML Refactoring**.

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

You can define short names for predefined blocks of code called *code templates*. The short names are displayed in the Content Completion window when the word at cursor position is a prefix of such a short name. If there is no prefix at cursor position (a whitespace precedes the cursor), all the code templates are listed.

Oxygen XML Editor plugin comes with numerous predefined code templates. You can also *define* your own code templates for any type of editor. For more details, see the [example for XSLT editor code templates](#).

To obtain the template list, you use the **Ctrl (Meta on Mac OS) + Space** content completion shortcut key, or the **Ctrl (Meta on Mac OS) + Shift + Space** code templates shortcut key. The first shortcut displays the code templates in the same *content completion list with elements from the schema of the document*. The second shortcut displays only the code templates and is the default shortcut of the action **Document > Content Completion > Show Code Templates**.

The syntax of the code templates allows you to use the following *editor variables*:

- **#{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. Can have one of the following values:
 - url - input is considered an URL. Oxygen XML Editor plugin checks that the URL is valid before passing it to the transformation;
 - password - input characters are hidden;
 - generic - the input is treated as generic text that requires no special handling;
 - relative_url - input is considered an URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing;



Note: You can use the `$ask` editor variable in file templates. In this case, Oxygen XML Editor plugin keeps an absolute URL.

- `combobox` - displays a dialog that contains a non-editable combo-box;
 - `editable_combobox` - displays a dialog that contains an editable combo-box;
 - `radio` - displays a dialog that contains radio buttons;
 - 'default-value' - optional parameter. Provides a default value in the input text box;
- Examples:**
- `#{ask('message')}` - Only the message displayed for the user is specified.
 - `#{ask('message', generic, 'default')}` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
 - `#{ask('message', password)}` - 'message' is displayed, the characters typed are masked with a circle symbol.
 - `#{ask('message', password, 'default')}` - same as before, the default value is 'default'.
 - `#{ask('message', url)}` - 'message' is displayed, the parameter type is URL.
 - `#{ask('message', url, 'default')}` - same as before, the default value is 'default'.
- **#{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; An 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;
- **#{cfd}** - 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 `frameworks` subfolder of the Oxygen XML Editor plugin installation folder;
- **#{pd}** - Current project folder as file path. Usually the current folder selected in the Project View;
- **#{oxygenInstallDir}** - Oxygen XML Editor plugin 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>.

To watch our video demonstration about code templates, go to http://oxygenxml.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:

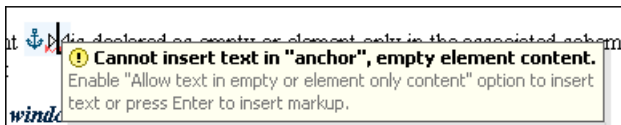


Figure 41: Editing in empty element warning

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.

The font size of the current WYSIWYG-like editor can be increased and decreased on the fly with the same actions as in the Text editor:

- **Ctrl (Meta on Mac OS) + NumPad + "+"** or **Ctrl (Meta on Mac OS) + NumPad + "-"** or **Ctrl (Meta on Mac OS) + Scroll Forward** - Increases font size.

- **Ctrl (Meta on Mac OS) + NumPad + "-"** or **Ctrl (Meta on Mac OS) + "-"** or **Ctrl (Meta on Mac OS) + Scroll Backwards** - Decreases font size.
- **Ctrl (Meta on Mac OS) + NumPad + 0** or **Ctrl (Meta on Mac OS) + 0** - Restores font size to *the size specified in Preferences*.

Removing the Text Content of the Current Element

You can remove the text content of the current element and keep only the markup with the action  **Remove Text** available on the submenu **Refactoring** of the contextual menu and on the toolbar **XML Refactoring**. 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 from DITA, Docbook and TEI document types) is turned off and you duplicate elements with existing IDs, the duplicates lose these IDs. If the previously mentioned option is active, when you duplicate content, Oxygen 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 ID's 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 plugin 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 42: Resizing a column in Oxygen XML Editor plugin 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 you position the mouse cursor above a column or in front of a row, without clicking, Oxygen XML Editor plugin only highlights the column or the row. When you make a selection, Oxygen XML Editor plugin

changes the cursor to  for row selection and to  for column 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 plugin 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 plugin, 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 plugin 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 plugin 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 only the content of a table, select a row or column and press either **Delete**, or **Backspace** on your keyboard.

To delete an entire row or column, use  **Delete a table row** or  **Delete a table column**.

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	<input type="text" value="c1"/>	number	<input type="text" value="1"/>	width	<input type="text" value="0.32*"/>	align	<input type="text" value=""/>	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	<input type="text" value="c2"/>	number	<input type="text" value="2"/>	width	<input type="text" value="1.49*"/>	align	<input type="text" value=""/>	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	<input type="text" value="c3"/>	number	<input type="text" value="3"/>	width	<input type="text" value="1.15*"/>	align	<input type="text" value=""/>	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	<input type="text" value="c4"/>	number	<input type="text" value="4"/>	width	<input type="text" value="0.4*"/>	align	<input type="text" value=""/>	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	<input type="text" value="c5"/>	number	<input type="text" value="5"/>	width	<input type="text" value="1.67*"/>	align	<input type="text" value=""/>	<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep

Horizontal Span		a3	a4	a5
<i>f1</i>	<i>f2</i>	<i>f3</i>	<i>f4</i>	<i>f5</i>
b1	b2	b3	b4	▶Vertical◀ Span
c1	Spans ▶Both◀ directions		c4	
d1			d4	d5

Figure 43: CALS table in Docbook

XHTML Table Layout

The HTML table model accepts both table and column widths. Oxygen XML Editor plugin uses the `width` attribute of the `table` element and the `col` element associated with each column. Oxygen XML Editor plugin displays the values in fixed units, proportional units, or percentages.

colspecs...

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 44: 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

colspecs...

column name	c1	number	1	width	0.32*	align		<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	c2	number	2	width	1.49*	align		<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	c3	number	3	width	1.15*	align		<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	c4	number	4	width	0.4*	align		<input type="checkbox"/> colsep	<input type="checkbox"/> rowsep
column name	c5	number	5	width	1.67*	align		<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	Spans ▸Both◄ directions		c4	
d1			d4	d5

Figure 45: 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 46: 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.

Sorting Content in Tables and List Items

Oxygen XML Editor plugin 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 an Entire Table

To sort an entire table either right click the table and select  **Sort**, or select the table and click  **Sort** on the main toolbar. Any of the two opens the **Sort** dialog box.

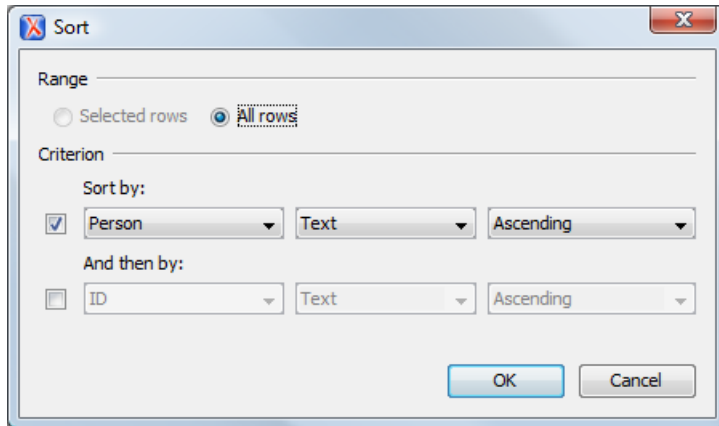



Figure 47: The "Sort" Dialog Box


This dialog box sets the range that is sorted and the sorting criterion. The range is selected automatically 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 **Criterion** section specifies the sorting criteria (at most three sorting criteria are available). Each sorting criteria is defined by:

- 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 set automatically 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 plugin recognizes multiple date formats like *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. In case you want to go back to the initial order of your content, press **Ctrl (Meta on Mac OS) + Z** on your keyboard.

 **Note:** The sorting support takes into account the value of the `xml:lang` attribute 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. Any of the two opens the **Sort** dialog box.

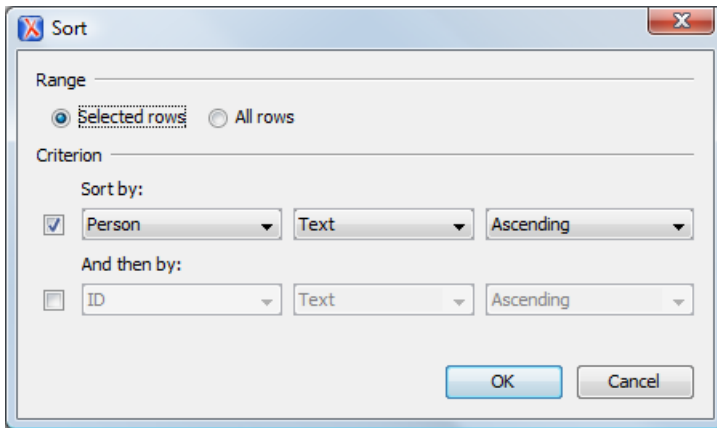



Figure 48: Sort Selected Rows

This dialog box sets the range that is sorted and the sorting criterion. The range is selected automatically depending on whether you sort an entire table or only a selection of its rows.


The **Criterion** section specifies the sorting criteria (at most three sorting criteria are available). Each sorting criteria is defined by:

- 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 set automatically 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 plugin recognizes multiple date formats like *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. In case you want to go back to the initial order of your content, press **Ctrl (Meta on Mac OS) + Z** on your keyboard.

 **Note:** The sorting support takes into account the value of the `xml:lang` attribute 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 items of each criterion and click **OK** to complete the sorting operation.

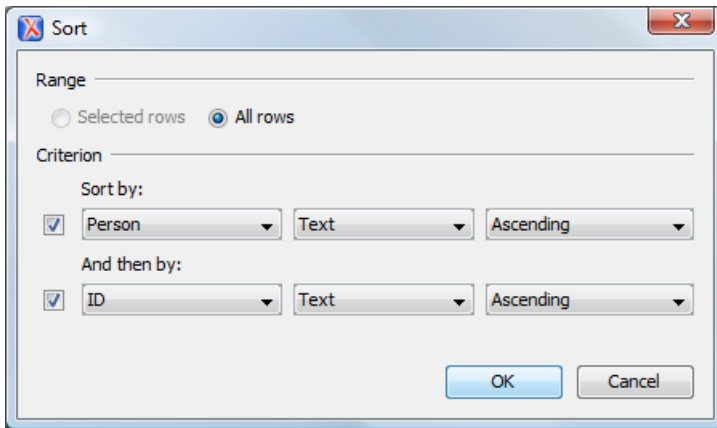


Figure 49: Sorting Based on Multiple Criteria

Sorting List Items

You can perform the sorting operation over list items of ordered and unordered lists.

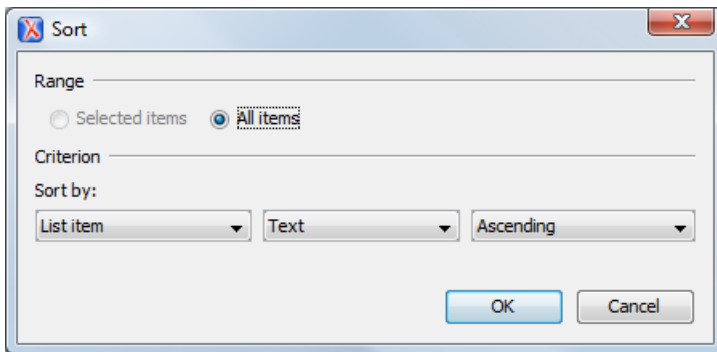




Figure 50: Sorting List Items

To sort the items in a list, either click  **Sort** on the main toolbar or right click the list and select  **Sort** from its contextual menu. The sorting mechanism works on an entire list and on a selection of list items as well.


 **Note:** The sorting support takes into account the value of the `xml:lang` attribute and sorts the content in a natural order.

Image Rendering

The **Author** editor and the output transformation process might render differently the images referenced in the XML document, since they use different rendering engines.

Table 2: 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	

Image Type	Support	Additional Information
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 plugin **Author** mode displays a warning message that contains the reason why this is happening. Possible causes:

- the image is too large. Enable *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 controls the rendered image;
- if you want to scale proportionally both the width and height of an image, use the *scale* attribute.



Note:

As a Java application, Oxygen XML Editor plugin uses Java Advanced Imaging which provides a pluggable support for new image types. In case you have an *ImageIO* library that supports additional image formats, just copy this library to `[oxygen install folder]/lib`.

Installing Java Advanced Imaging(JAI) Image I/O Tools plug-in

Follow this procedure:

1. Start Oxygen XML Editor plugin and open the **Help > About** dialog. Open the **Installation Details** dialog, **Configuration** tab and look for *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.

Mac OS X Workaround


There is no native implementation of JAI Image I/O for Mac OS X 10.5 and later. However, the JAI Image I/O has a Java implementation fallback which also works on Mac 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/lib` directory create a directory named `endorsed` e.g. `Oxygen/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/lib/endorsed` directory.
4. Restart the application and the JAI Image I/O support will be up and running.

Customize Oxygen XML Editor plugin to Render CGM Images (Experimental Support)

Oxygen XML Editor plugin provides experimental support for CGM 1.0 images.

 **Attention:** Image hotspots are not supported.

Since it 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 `OXYGEN_INSTALL_DIR\lib` directory.
3. Open `OXYGEN_PLUGIN_DIR/META-INF/MANIFEST.MF` and add a reference to the JAR library in the `Bundle-ClassPath` entry.
4. Restart the application.

Customize Oxygen XML Editor plugin to Render PDF Images (Experimental Support)

Oxygen XML Editor plugin 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. Download the `pdfbox-1.8.3-src.zip` from <http://pdfbox.apache.org/>.
2. Unpack the ZIP archive and copy the `pdfbox-app-1.8.3.jar` into `OXYGEN_INSTALL_DIR\lib` directory.
3. Open `OXYGEN_PLUGIN_DIR/META-INF/MANIFEST.MF` and add a reference to the JAR library in the `Bundle-ClassPath` entry.
4. Restart the application.

Customize Oxygen XML Editor plugin to Render EPS and AI Images

Most EPS and AI image files include a preview picture of the content. Oxygen XML Editor plugin 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 plugin 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 plugin 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 plugin 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 plugin 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"/>
```
- Copy the image from other application (like an image editor) and paste it in your document. Oxygen XML Editor plugin prompts you to first save it. After saving the image to a file, a reference to that file path is inserted at the drop 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.

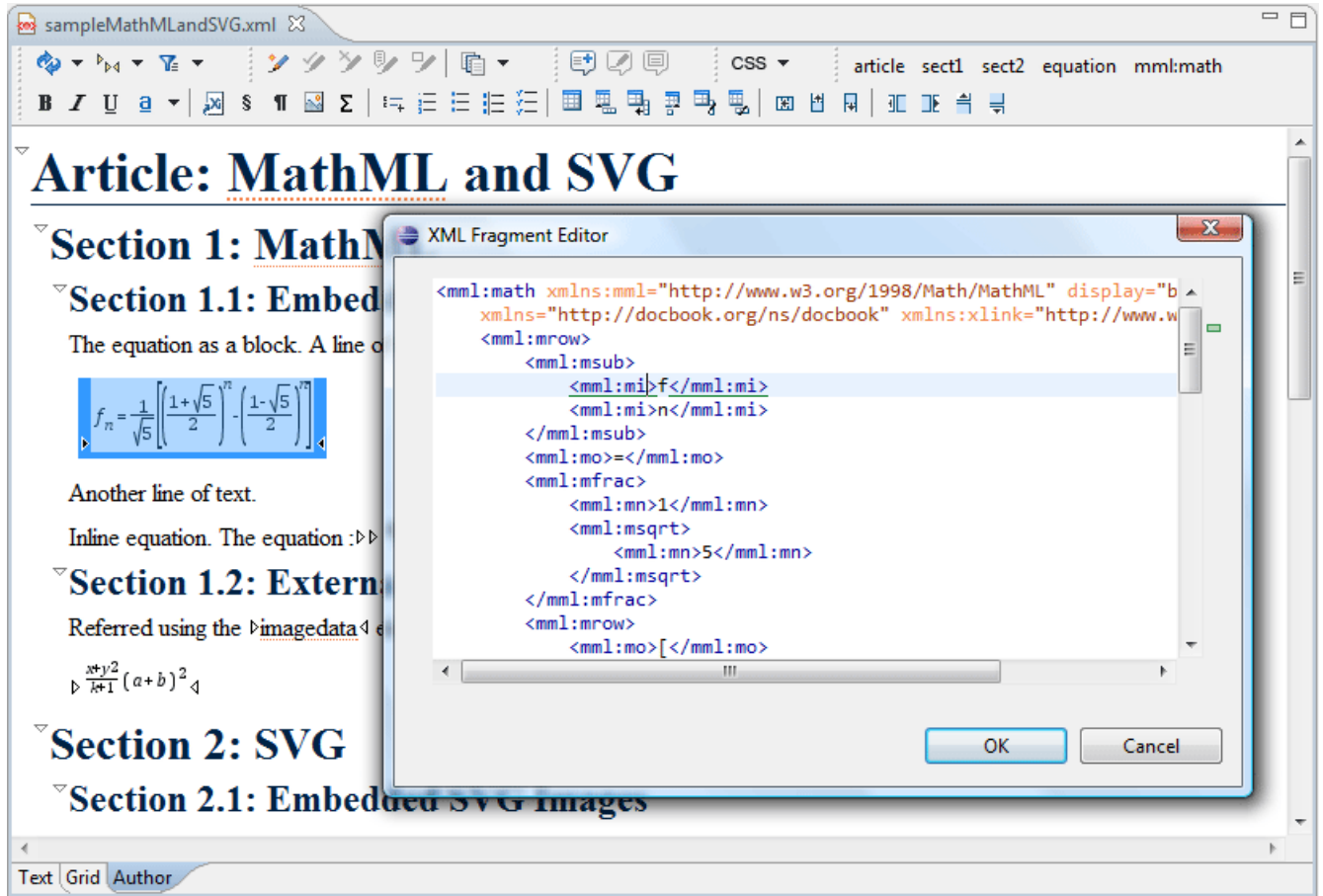




Figure 51: The default *MathML* editor

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.

For refreshing the content of the referred resources you can use the  **Refresh references** action. However, this action will not refresh the expanded external entities, to refresh those you will need to use the **Reload** action.

Validation and Error Presenting

Automatic validation as well as validate on request operations are available while editing documents in the Author editor. A detailed description of the document validation process and its configuration is described in section [Validating Documents](#).



Figure 52: Error presenting in Oxygen XML Editor plugin Author editor

A fragment with a validation error or warning will be marked by underlining the error region with a red color. The same will happen for a validation warning, only the color will be yellow instead of red.

- The top area - a success validation indicator that will turn green in case the validation succeeded or red otherwise.
- The middle area - the errors markers are depicted in red. The number of markers shown can be limited by modifying the setting **Window > Preferences > oXygen XML Editor > Editor > Document checking > Maximum number of errors reported per document**.

Status messages from every validation action are logged into the *Console view*.

Whitespace Handling

There are several major aspects of white-space handling in Oxygen XML Editor plugin which are important in the following cases:

- when opening documents;
- when switching from other editing mode to **Author** mode;
- when saving documents in **Author** mode;
- when switching from Author mode to another one.
- **Open documents** - When deciding if the white-spaces from a text node are to be preserved, normalized or stripped, the following rules apply:
 - If the text node is inside an element context where the `xml:space="preserve"` is set then the white-spaces are preserved;
 - If the CSS property `white-space` is set to `pre` for the node style then the white-spaces are preserved;
 - If the text node contains other non-white-space characters then the white-spaces are normalized;
 - If the text node contains only white-spaces:
 - If the node has a parent element with the CSS `display` property set to `inline` then the white-spaces are normalized;
 - If the left or right sibling is an element with the CSS `display` property set to `inline` then the white-spaces are normalized;
 - If one of its ancestors is an element with the CSS `display` property set to `table` then the white-spaces are striped;
 - Otherwise the white-spaces are ignored.

- **Save documents** - The Author editor will try to format and indent the document while following the white-space handling rules:
 - If text nodes are inside an element context where the `xml:space="preserve"` is set then the white-spaces are written without modifications;
 - If the CSS property `white-space` is set to `pre` for the node style then the white-spaces are written without any changes;
 - In other cases the text nodes are wrapped.

Also, when formatting and indenting an element that is not in a space-preserve context, additional line separators and white-spaces are added as follows:

- Before a text node that starts with a white-space;
 - After a text node that ends with a white-space;
 - Before and after CSS `block` nodes;
 - If the current node has an ancestor that is a CSS `table` element.
- **Editing documents** - You cannot insert consecutive space characters in any text nodes. Line breaks are permitted only in space-preserve elements. Tabs are marked in the space-preserve elements with a little marker.

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 plugin project file that will be shared by all content authors.
2. Set your own preferences in the following panels of the **Preferences** dialog: **Editor / Format** and **Editor / Format / XML**.
3. Save the preferences of these two panels in the Oxygen XML Editor plugin 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.

Managing Changes

You can review the changes you or other authors made and then accept or reject them using the **Track Changes** toolbar




buttons , 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 when navigating changes;
-  **Highlight** - enables the *Highlight tool*;

- **Colors** - opens the colors palette of the *Highlight tool*;
- **Stop Highlighting** - disables the *Highlight tool*;
-  **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

Four specialized actions 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 a drop-down list in the **Author 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://oxygenxml.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.

What do you want to do?

- *Keep tracking of inserted content;*
- *Keep tracking of deleted characters;*
- *Keep tracking of deleted content;*
- *Keep tracking of copied content;*
- *Keep tracking of pasted content;*
- *Keep tracking of 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 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 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. Some of the limitations are listed below:

1. 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.
2. Surrounding a selection with a certain element is not (yet) recorded as a change.

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	Element Start Tag	Element End Tag	Element Attributes
Insertion	<?oxy_insert_start?>	<?oxy_insert_end?>	author, timestamp
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 is intersecting with another, the `mid` attribute is used to correctly identify start tags and end tags.

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"?>
```

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 **Author 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 also the **XHTML** 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 53: Change Tracking in Author Mode

When hovering 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. The following table offers you a detailed view of the tracked and untracked changes:

Tracked Changes	Untracked Changes
Inserting, deleting content (text or elements)	Performing a Split operation
Drag and drop content (text or elements)	Performing a Surround with operation
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	
Deleting element tags	

Tracked Changes	Untracked Changes
Editing attributes	

If the selection in the **Author** contains track 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 instructions are examples of storing *insert* and *delete* changes in the document:

- `<?oxy_insert_start author="John Doe" timestamp="20090408T164459+0300"?>all<?oxy_insert_end?>`
- `<?oxy_delete author="John Doe" timestamp="20090508T164459+0300" content="belong"?>`



Note: The **Outline** view is synchronized with the **Track Changes**. Deleted content is rendered with a strike through in the **Outline** view.

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 **Author Review** toolbar and on the **Review** submenu of the contextual menu of 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 plugin presents the tracked changes in DITA conrefs and XInclude fragments.




Managing Comments

A comment is marked in the **Author** mode with a background that is configured for each user name.


The screenshot shows a document titled "Spring Flowers" with a comment box overlaid on the text. The comment box contains the text: "This list of spring flowers is too long. Also it should be organized as an itemized list, the inline enumeration is not readable." The comment box also shows the author's name "sorin" and the timestamp "Fri Sep 03 04:37 PM 2010".


Figure 54: Manage Comments in Author Editor

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 on the Author toolbar;
-  **Edit Comment...** - allows you to change an existing content. The action is available both on 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 on 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** set 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 plugin 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 your 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 (Meta on Mac OS)+A** 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 plugin 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 plugin assigns an individual color for the callouts depending on the user who is editing the document. To customize the list of these colors, go to **Options > Preferences > 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 plugin 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 plugin 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, go to **Options > Preferences > Editor > Edit Modes > Author > Review > Callouts**. Enable the following options:

- **Comments** - Oxygen XML Editor plugin displays comment callouts when you insert a comment. You can use two types of comments in Oxygen XML Editor plugin:
 - 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 the content of a comment callout and display the date and time of its insertion, go to **Options > Preferences > Editor > Edit Modes > Author > Review > Callouts** and enable **Show review time**;

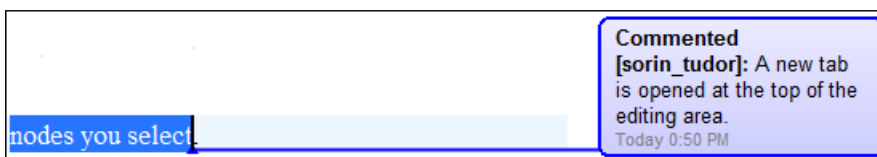


Figure 55: Comment Callouts

- **Track Changes deletions** - Oxygen XML Editor plugin 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, go to **Options > Preferences > Editor > Edit Modes > Author > Review > Callouts** and enable **Show review time** and **Show deleted content in callout**;

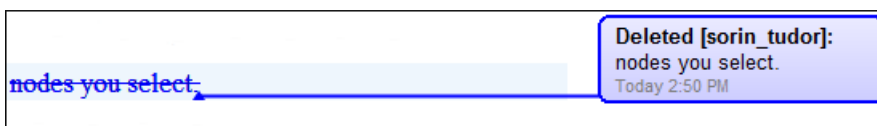


Figure 56: Deletion Callouts

- **Track Changes insertions** - Oxygen XML Editor plugin 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. To do this, go to **Options > Preferences > Editor > Edit Modes > Author > Review > Callouts** and enable **Show review time** and **Show inserted content in callout**.

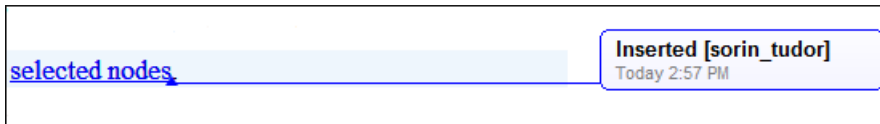


Figure 57: Insertion Callouts

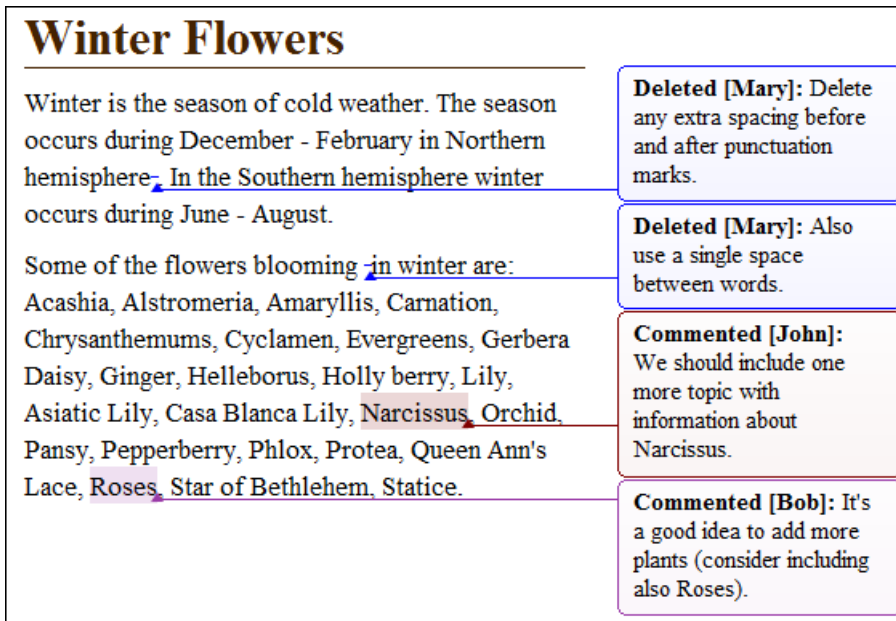







Figure 58: Multiple Authors Callouts

 **Note:** Oxygen XML Editor plugin displays callouts only if  **View All Changes/Comments** or **View Only Changes/Comments** by is selected. Oxygen XML Editor plugin 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 plugin 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.

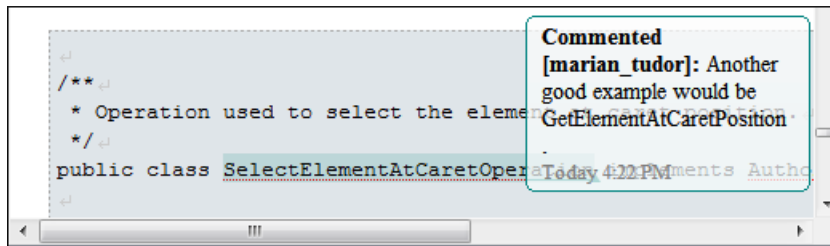



Figure 59: Transparent Callout



Note: Oxygen XML Editor plugin 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 **Author Review** toolbar;
- **Edit Reference** - in case 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** - in case 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.

When you print a document from Oxygen XML Editor plugin, 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://oxygenxml.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.

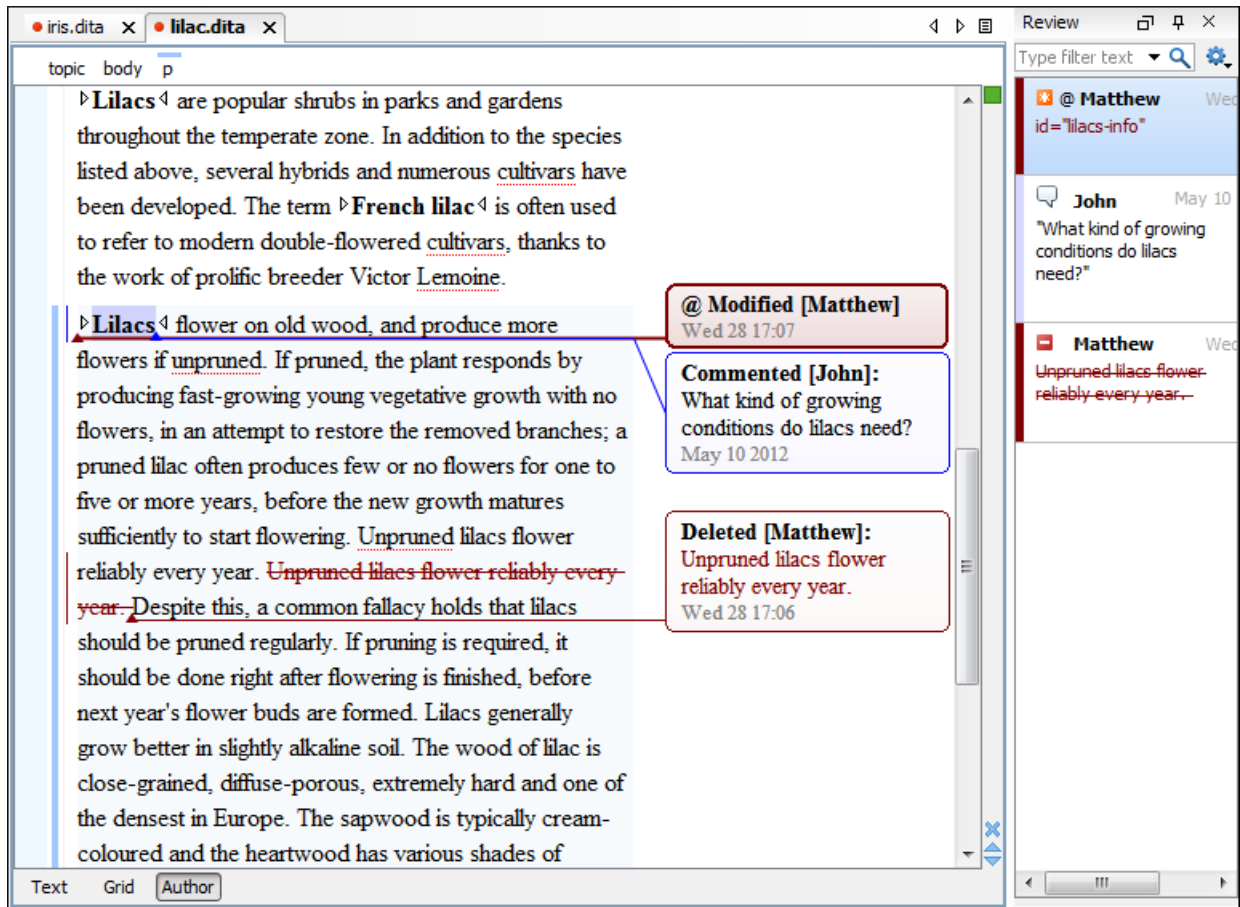



Figure 60: The Review View

To activate the **Review** view, do one of the following:

- click the  **Manage reviews** button on the **Author 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** - this action is available for highlights and comments presented in the **Review** view. Use this action to remove these highlights or comments from your document;
- **Accept** - this action is available for inserted and deleted content presented in the **Review** view. Use this action to accept the changes in your document;
- **Reject** - this action is 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 in a document that has the same color;
- **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.

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 plugin 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

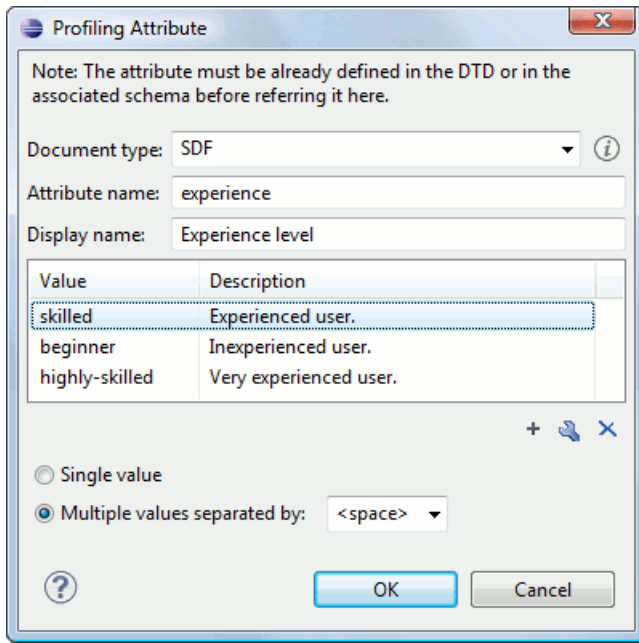


Note: To ensure the validity of the document, the attribute must be already defined in the document DTD or schema before referring it here.

To create custom profiling attributes for a specific document type, follow these steps:

1. Open the **Profiling/Conditional Text** preferences page from **Window > Preferences > oXygen > Editor > Edit modes > Author > Profiling / Conditional Text**.
2. In the **Profiling Attributes** area, press the **New** button.

The following dialog is opened:



3. Fill-in the dialog as follows:

- a) Choose the document type on which the profiling attribute is applied. * and ? are 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) Set the attribute name.
- c) Set a display name. This field is optional, being used only as a more descriptive rendering in application's profiling dialogs.
- d) Use the **New**, **Edit**, **Delete** buttons to add, edit and delete possible values of the attribute. Each attribute value can have a description.
- e) Choose whether the attribute accepts a single value (**Single value** option checked) or multiple values. Multiple values can be separated by a default delimiter (*space, comma, semicolon*), or a custom one, that 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.

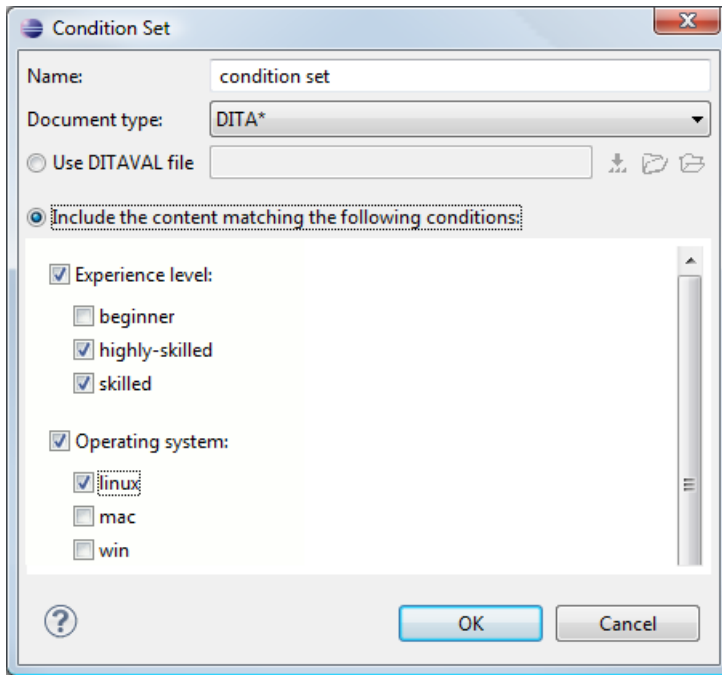
Create Profiling Condition Sets

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 **Profiling/Conditional Text** preferences page from **Window > Preferences > oXygen > Editor > Edit modes > Author > Profiling / Conditional Text**.
2. In the **Profiling Condition Sets** area, press the **New** button.

The following dialog is opened:



3. Fill-in the dialog as follows:

a) Type the condition set's name.


If you want the Profiling Condition Set to refer 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.

c) Define the combination of attribute values by ticking the appropriate checkboxes.

4. Click **OK**.

5. Click **Apply** to save the condition set. All saved profiling condition sets are available in the  **Profiling / Conditional Text toolbar 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. 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:

▼ 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]

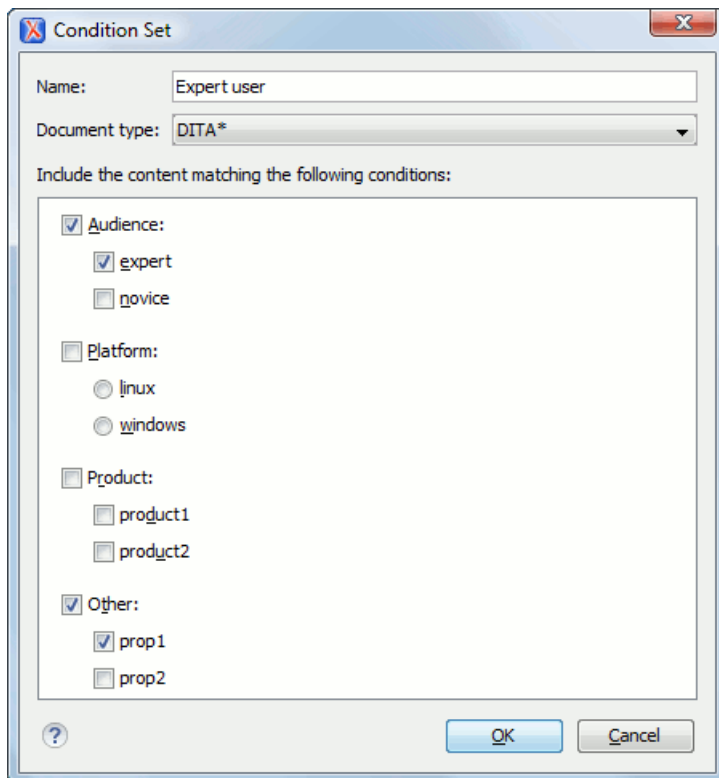
Step 2
Place newspaper, cardboard, or a drop-cloth on the garage floor. Audience [expert]

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]

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 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, 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 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 plugin 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 Attributes** - Enable this option to turn on conditional text markers. They are displayed at the end of conditional text block, as a list of attribute name and their currently set values.
- The list of all profiling condition sets that match the current document type. Click on a condition set entry to activate it.
-  **Configure Profiling Condition Sets...** - Link to the *Profiling / Conditional Text* preference page, where you can manage profiling attributes and profiling condition sets.

Smart Paste Support

The *Smart Paste* capability was developed to help authors copy content from various sources (like web pages or office-type documents) and paste it into DITA, TEI, Docbook and XHTML documents. Oxygen XML Editor plugin eases this process by keeping the original text styling (like bold, italics) and formatting (like lists, tables, paragraphs), while providing assistance to obtain a valid structured document.

The Oxygen XML Editor plugin *Smart Paste* support encapsulates the following capabilities:

- The conversion of the copied content into valid DITA, Docbook, TEI and XHTML fragments:

Styled content can be inserted in the Author editor by copying or dragging it from:

- Office-type applications (**Microsoft Word** and **Microsoft Excel**, **OpenOffice.org Writer** and **OpenOffice.org Calc**);
- web browsers (like **Mozilla Firefox** or **Microsoft Internet Explorer**);
- the **Data Source Explorer** view (where resources are available from WebDAV or CMS servers).

The styles and general layout of the copied content like: sections with headings, tables, list items, bold, and italic text, hyperlinks, are preserved by the paste operation by transforming them to the equivalent XML markup of the target document type. This is available by default in the following *predefined document types*: *DITA*, *DocBook 4*, *DocBook 5*, *TEI 4*, *TEI 5*, *XHTML*.

This support is enabled by default, but you can disable it through the *Convert external content on paste* option, available in the **Schema Aware** preferences.

- Inserting the converted fragment at the correct location in the document.

This capability is controlled by the *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://oxygenxml.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 plugin 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 plugin 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.

To watch our video demonstration about the bidirectional text support in the **Author** mode, go to http://oxygenxml.com/demo/BIDI_Support.html.

Controlling the Text Direction Using XML Markup

Oxygen XML Editor plugin Supports the following CSS properties:

Table 3: CSS Properties Controlling Text Direction

<code>direction</code>	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 direction property is inherited from the parent element).
<code>unicodeBidi</code>	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 plugin recognizes the `dir` attribute on any XML document. The supported values are:

<code>ltr</code>	The text from the current element is Left to Right, embedded.
<code>rtl</code>	The text from the current element is Right to Left, embedded.
<code>lro</code>	The text from the current element is Left to Right, embedded.
<code>rlo</code>	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 4: 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 **Symbols** toolbar action. 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 plugin offers the support for bi-directional text in all the side views (**Outline** view, **Attributes** view and so on) and text fields.

Chapter

5

Editing Documents

Topics:

- [Working with Unicode](#)
- [Creating, Opening, and Closing Documents](#)
- [Grouping Documents in XML Projects](#)
- [Editing XML Documents](#)
- [Editing XSLT Stylesheets](#)
- [Editing XML Schemas](#)
- [Editing XQuery Documents](#)
- [Editing WSDL Documents](#)
- [Editing CSS Stylesheets](#)
- [Editing Relax NG Schemas](#)
- [Editing NVDL Schemas](#)
- [Editing JSON Documents](#)
- [Editing StratML Documents](#)
- [Editing JavaScript Documents](#)
- [Editing XProc Scripts](#)
- [Editing Schematron Schemas](#)
- [Spell Checking](#)
- [Handling Read-Only Files](#)
- [Associating a File Extension with Oxygen XML Editor plugin](#)
- [Terms](#)

This chapter explains the editor types available in Oxygen XML Editor plugin 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 plugin 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 plugin XML Editor uses 16bit characters covering the Unicode Character set.



Note: Oxygen XML Editor plugin 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 plugin receives the encoding of the document from the Eclipse platform. This encoding is then used to instruct the Java Encoder to load support for and to save the document using the specified code chart.

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.

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 plugin, 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.

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.

Oxygen XML Editor plugin Wizard for New Document

The **New Document** wizard only creates a skeleton document. It contains the document prolog, a root element, and possibly other child elements depending on the options specific for each schema type. To generate full and valid XML instance documents based on an XML Schema, use the [XML instance generation tool](#).

The Oxygen XML Editor plugin plugin installs a series of Eclipse wizards for easy creation of documents. If you use these wizards, Oxygen XML Editor plugin automatically completes the following details:

- the system ID, or schema location of a new XML document;
 - the minimal markup of a DocBook article, or the namespace declarations of a Relax NG schema.
1. To create a document, either select **File > New > -> Other > Ctrl (Meta on Mac OS) + N > oXygen**, or click the  **New** button on the toolbar.
The **New** wizard is displayed.

2. Select a document type.
3. Click the **Next** button.

For example if XML was selected the **Create an XML Document** wizard is started.

The **Create an XML Document** dialog box enables definition of an XML Document Prolog using the system identifier of an XML Schema, DTD, Relax NG (full or compact syntax) schema, or NVDL (Namespace-based Validation Dispatching Language) schema. As not all XML documents are required to have a Prolog, you can choose to skip this step by clicking **OK**. If the prolog is required, complete the fields as described in the next step.

4. Type a name for the new document and press the **Next** button.
5. If you select **Customize**, Oxygen XML Editor plugin opens the following dialog box. You can customize different options depending on the document type you select.

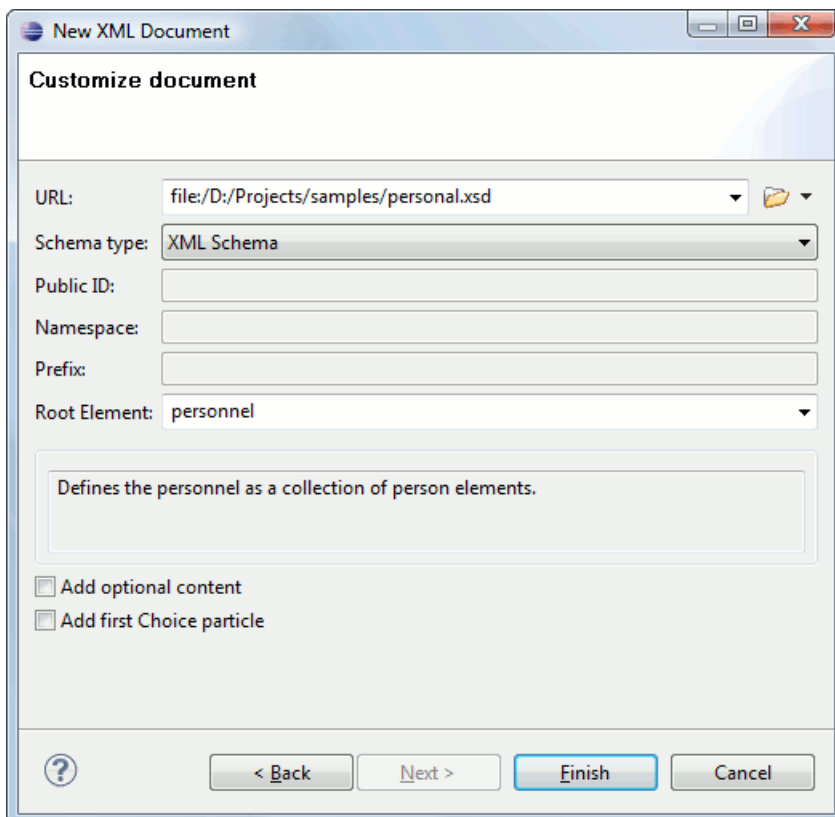


Figure 61: New XML Document Dialog Box

- **Schema URL** - specifies the path to the schema file. When you select a file, Oxygen XML Editor plugin 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 plugin generates the first element of an *xs:choice* schema element in the skeleton XML document. Oxygen XML Editor plugin creates this document in a new editor panel when you click **OK**.

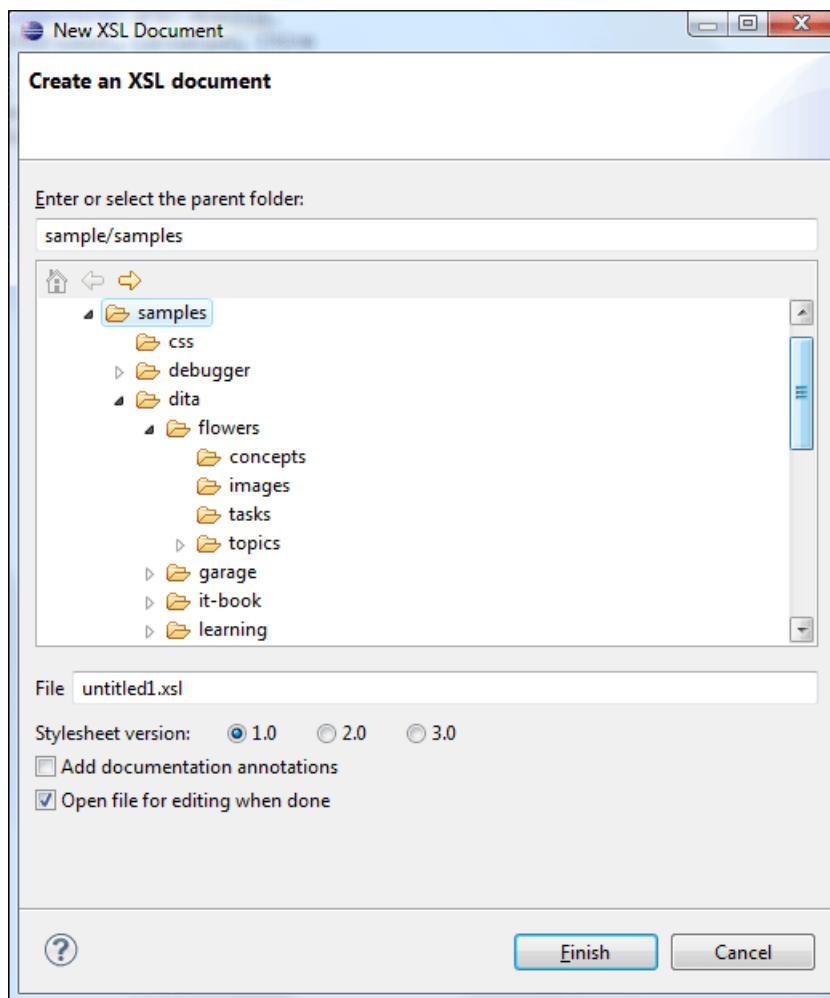


Figure 62: New XSL Document Dialog Box

- **Stylesheet version** - allows you to select the Stylesheet version number. You can select from: 1.0, 2.0, and 3.0;
- **Add documentation annotations** - adds annotation for XSL components;
- **Open file for editing when done** - when you press **Finish**, Oxygen XML Editor plugin opens the newly created file.

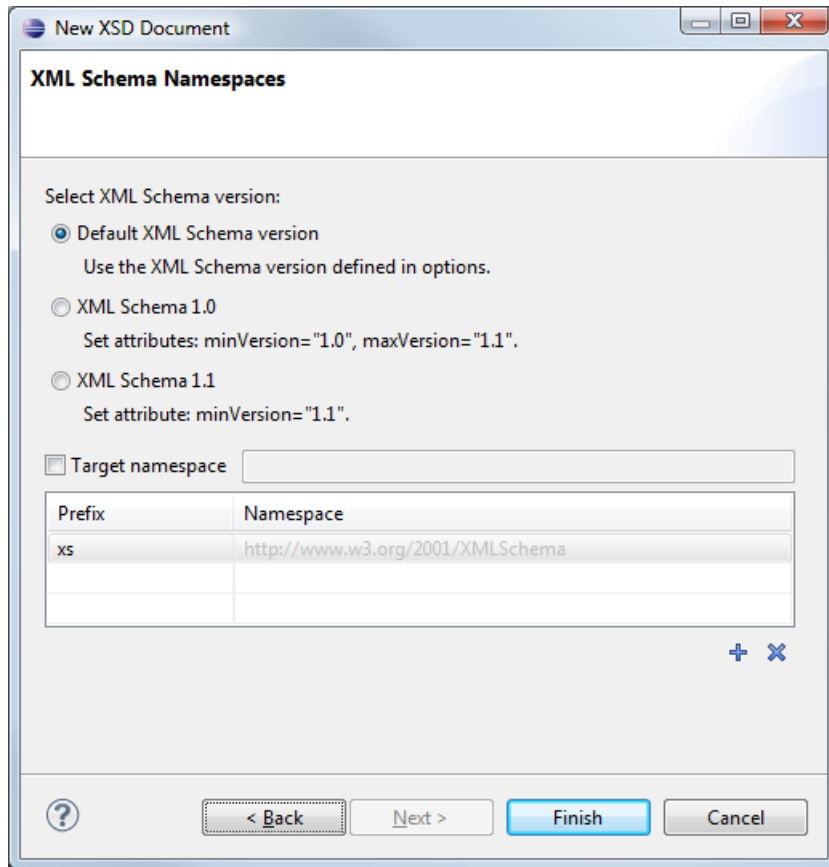



Figure 63: New XML Schema Document Dialog Box

- **Default XML Schema version** - Uses the XML Schema version defined in the **Options > Preferences > 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. To manage table information, use 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](#).

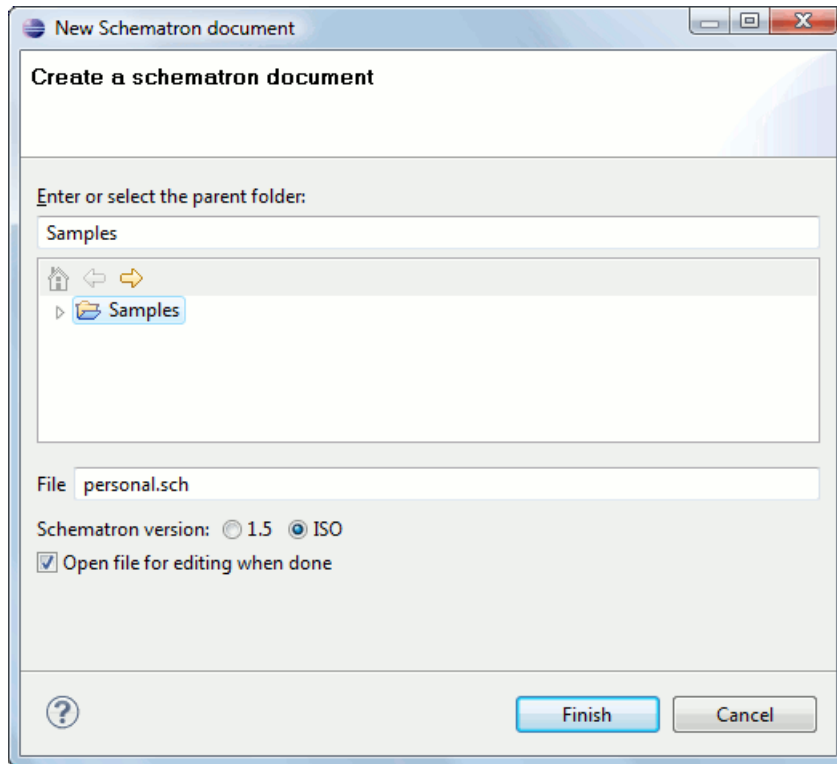


Figure 64: New Schematron Document Dialog Box

- **Schematron version** - specifies the Schematron version. Possible options: 1.5 and ISO.

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 - The user can add template files in the `templates` folder of the Oxygen XML Editor plugin install directory. Also in the option page **Window > Preferences > oXygen > Editor > Templates > Document Templates** can be specified a custom templates folder to be scanned.

1. Go to menu **File > New > Other > oXygen > New From Templates**.
2. Select a document type.
3. Type a name for the new document and press the **Next** button.
4. 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 are able to build new documents rapidly, based on the same characteristics (file type, prolog, root element, existing content). Oxygen XML Editor plugin offers a rich set of templates for a number of XML applications. You may also create your own templates from **Options > Windows > Preferences > oXygen > Editor > Templates > Document Templates** and share them with others.

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;**
- **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 All** - saves all open documents.


Opening and Saving Remote Documents via FTP/SFTP

Oxygen XML Editor plugin supports editing remote files, using the FTP, SFTP protocols. You can edit remote files in the same way you edit local files.

You can open one or more remote files in *the [Open using FTP/SFTP dialog](#)*

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 plugin and the HTTP / WebDAV server is compressed using the GZIP algorithm.

The current *[WebDAV Connection](#)* details can be saved using the  button and then used in the *Data Source Explorer* view.

The Open Using FTP/SFTP Dialog

To open the **Open using FTP/SFTP** dialog, go to **File > Open URL ...** or click the  **Open URL ...** toolbar button.

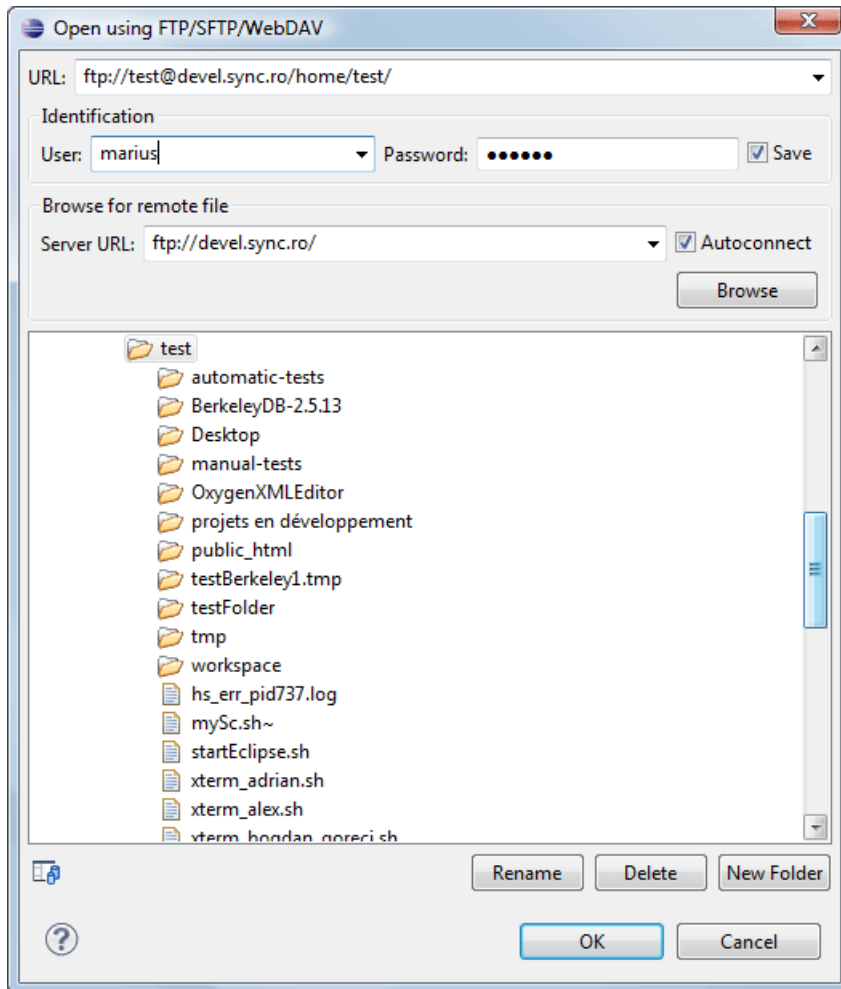


Figure 65: Open URL dialog

The displayed dialog is composed of several parts:

- 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 `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.

- The *Identification* section contains the access credentials. If you want 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 into 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.

- The *Browse for remote file* section contains the server combo and the **Autoconnect** check box. In the server combo you can specify the protocol, the server host name or server IP.

**Tip:**

Server URLs

When accessing a FTP server, you need to specify only the protocol and the host, like: ftp://server.com, or if using a nonstandard port: ftp://server.com:7800/.

By pressing the **Browse** button the directory listing will be shown in the component below. When **Autoconnect** is selected then at every time the dialog is shown, the browse action will be performed.

- The tree view of the documents stored on the server. You can browse the directories, and make multiple selections. Additionally, you may use the **Rename**, **Delete**, and **New Folder** to manage the file repository.

The file names are sorted in a case-insensitive way.

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 file browser dialog by right-clicking on a tree node and selecting the *Change permissions* menu item.

The usual Unix file permissions *Read*, *Write* and *Execute* are granted or denied in this dialog for the file owner, owner group and the rest of the users. The permission's aggregate number 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 plugin 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 plugin 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 plugin. This means that Oxygen XML Editor plugin 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 plugin cannot connect to an HTTPS-capable server, most likely there is no certificate set in the *Java Runtime Environment (JRE)* that oXygen 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 plugin.

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.

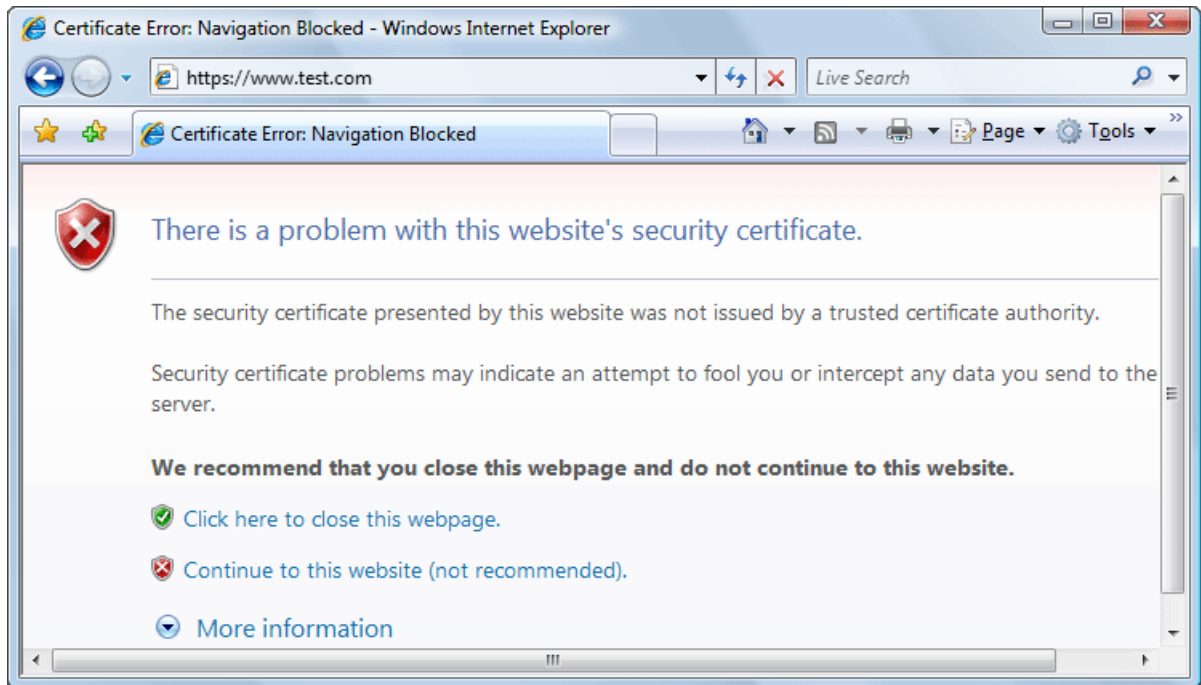


Figure 66: Security alert - untrusted certificate

- b) Go to menu **Tools > Internet Options**.
Internet Options dialog is opened.
 - c) Select **Security** tab.
 - d) Select **Trusted sites** icon.
 - e) Press **Sites** button.
This will open **Trusted sites** dialog.
 - f) Add repository URL to **Websites** list.
 - g) Close **Trusted sites** dialog and **Internet Options** dialog.
 - 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 containing **View certificates** link is displayed.
 - k) Click on **View certificates** link.
Certificate dialog is displayed.
 - l) Select **Details** tab of **Certificate** dialog.
 - 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 plugin.
 - a) Open a text-mode console with administrative rights.
 - b) Go to the `lib/security` directory of the JRE running Oxygen XML Editor plugin. You find the home directory of the JRE in the `java.home` property that is displayed in the **About** dialog. 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 plugin accept a certificate even if it is invalid, go to the **Options > Preferences > Connection settings > HTTP(S)/WebDAV** preferences page and enable the **Automatically accept a security certificate, even if invalid** option.

3. Restart Oxygen XML Editor plugin.

Opening the Current Document in System Application

To open the currently edited document in the associated system application, use the **Open in Browser/System Application** action available on the **XML > File** menu and also on the **Document** toolbar. The action is enabled when the current document has the file, FTP, HTTP or SFTP protocol.

Closing Documents

To close open documents, use one of the following methods:

- Go to menu **File > Close (Ctrl (Meta on Mac OS)+F4)**: Closes only the selected tab. All other tab instances remain opened.
- Go to menu **File > Close All (Ctrl (Meta on Mac OS)+Shift+F4)**: 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 Mac 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 > Other... > Editor Properties**.



To copy a value from the **Editor 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.

Creating a New Project






The tree structure occupies most of the view area. In the upper left side of the view, there is a drop-down list that holds all recently used projects and project management actions:

-  **Open Project ... (Ctrl (Meta on Mac OS)+F2)** - Opens an existing project. An alternate way to open a project is to drop an Oxygen XML Editor plugin XPR project file from the file explorer in the **Project panel**.
-  **New Project** - Creates a new, empty project.

The files are organized in an XML project usually as a collection of folders. They are created and deleted with the usual Eclipse actions.

Creating New Project Items

A series of actions are available in the contextual menu:

- **New >  File** - Creates a new file and adds it to the project structure.
- ** Add Folder** - Adds a link to a physical folder, whose name and content mirror a real folder existing in the file system on disk. The icon of this action is different on Mac OS X () as the standard folder icon on Mac OS X is not the usual one from Windows and Unix/Linux systems.
- **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 project, after closing the current project and all open files.

Managing Project Content


Validate Files

The currently selected files associated to the Oxygen XML Editor plugin in the **Package Explorer** view can be validated against a schema of type Schematron, XML Schema, Relax NG, NVDL, or a combination of the later with Schematron with one of the following contextual menu actions:

- **Validate** available on the **Batch Validation** submenu of the contextual menu of the **Package Explorer** view.
- **Validate with ...** available on the **Batch Validation** submenu of the contextual menu of the **Package Explorer** view.

Applying Transformation Scenarios


The currently selected files associated to the Oxygen XML Editor plugin in the **Package Explorer** view can be transformed in one step with one of the actions **Apply Transformation**, **Configure Transformation ...** and **Transform with...** available on the **Transformation** sub-menu of the right-click menu of the **Package Explorer** view. This, together with the logical folder support of the project allows you to group your files and transform them very easily.

If the resources from a linked folder in the project have been changed outside the view, you can refresh the content of the folder by using the  **Refresh** action from the contextual menu. The action is also performed when selecting the linked resource and pressing F5 key

You can also use drag and drop to arrange the files in logical folders. Also, dragging and dropping files from the project tree to the editor area results in the files being opened.

Other Context-Dependent Actions

Many of the actions available in the **Project** view are grouped in a contextual menu. This menu is displayed after selecting a file or folder and then pressing right-click (or Ctrl+Click on Mac OS X)

- **Open with** - Open selected file with one of internal tools: , , , WSDL/SOAP Analyzer, , .
- **Open All Files** - Opens in the editor view all files contained by the selected resources.
-  **Refresh** - Refreshes the content of the **Project** view;



Create an Oxygen XML Editor plugin XML Project

To create an XML project in Oxygen XML Editor plugin, follow these steps:

1. Go to menu **File > New > Ctrl (Meta on Mac OS) + N > XML Project**. The **New XML Project** wizard is displayed.
2. Type a name for the new project.
3. Click the **Next** button.
4. Select other Eclipse projects that you want to reference in the new project.
5. Click the **Finish** button.

Moving/Renaming Resources in the Navigator 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 right click the file or directory and select the **Refactoring > Move resource** action from its contextual menu, or use the usual  **Cut** and  **Paste** actions. Oxygen XML Editor plugin presents the **Move** dialog if you used the drag and drop or the **Cut/Paste** actions. The **Move resource** dialog is presented in case you used the refactoring actions. The following fields are available:

- **Destination** - available in the **Move resource** dialog only. 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, right click a file or a directory and select the **Refactoring > Rename resource** action from its contextual menu. The **Rename resource** dialog 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*.

In case the **Update references of the moved resource(s)** option is enabled, a **Preview** option which opens the **Preview** dialog is available for both actions. The **Preview** dialog presents a list with the resources that are updated.

 **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 case the references of a moved or a renamed resource can not be update. One case is when a resource is resolved through an XML catalog. Another problem can appear when the path to the moved/renamed resource contains entities. For these cases, Oxygen XML Editor plugin displays a warning dialog.

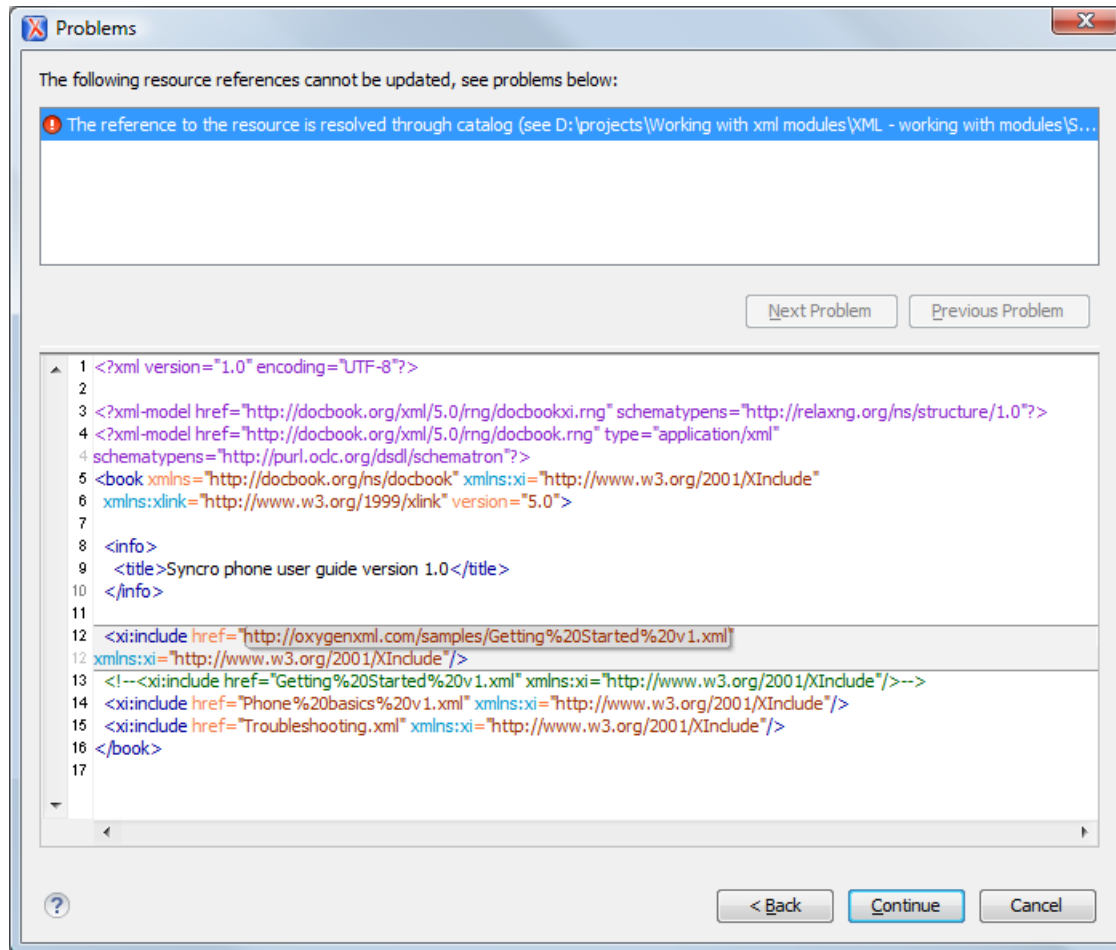


Figure 67: Problems Dialog

Defining Master Files at Project Level

This chapter details the **Master Files** support available in Oxygen XML Editor plugin.

The **Master Files** support helps you simplify the configuration and development of XML projects. A master file generally means the root of an import/include tree of modules.

Introduction

Oxygen XML Editor plugin allows you to define master files at project level. These master files are automatically used by Oxygen XML Editor plugin to determine the context for operations like validation, content-completion, refactoring or search for XML, XSD, XSL, WSDL, and RNG modules. Oxygen XML Editor plugin 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 plugin automatically identifies the master files which include that module and validates all of them;
- the **Content Completion Assistant** presents all the components collected starting from the master files towards the modules they include;
- the **Outline** view displays all the components defined in the master files hierarchy;
- the master files defined for the current module determines the *scope of the search and refactor actions*. Oxygen XML Editor plugin performs the search and refactor actions in the context that the master files determine, improving the speed of execution.

Enabling the Master Files Support

Oxygen XML Editor plugin stores the master files in a folder located in the **Navigator** view, as the first child of the project root. The **Master Files** support is disabled by default. To enable the **Master Files** support, use the **Enable Master Files Support** action from the contextual menu of the project itself. Oxygen XML Editor plugin allows you to enable/disable the **Master Files** support for each project you are working on.

Detecting Master Files

Oxygen XML Editor plugin 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 these options display the **Detect Master Files** wizard dialog. In the first panel you can select what type of master files you want Oxygen XML Editor plugin to detect. In the following panel the detected master files are presented in a tree like fashion. The resources are grouped in 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. No need to be set as master files.

To set the master files you can enable their check-boxes. Oxygen XML Editor plugin marks all the children of a master file as modules. Modules are rendered in gray and their tool-tip presents a list with their master files. A module can be accessed from more than one master file.

The master files already defined in the project are marked automatically 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 third 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 mark automatically all master files. This action sets as master files all the resources from the **Possible Master Files** category and the first resource of each **Cycle**.



Tip: We recommend you to add only top-level files (files that are 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.

Adding/Removing a Master File

The **Master Files** directory contains only logic 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;
- drag and drop files in the **Master Files** directory;
- from the contextual menu of the  **Resource Hierarchy Dependencies** view, using the  **Add to Master Files** action.

You can view the master files for the current edited resource in the **Editor Properties** view.

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 plugin.

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

Oxygen XML Editor plugin uses the following search pattern when it tries to detect an XML schema:

- in the *validation scenario* associated with the document;
- in the validation scenario associated with the document type (if defined).
- specified in the document;



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);

- detected from the document type that matches the edited document - 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.

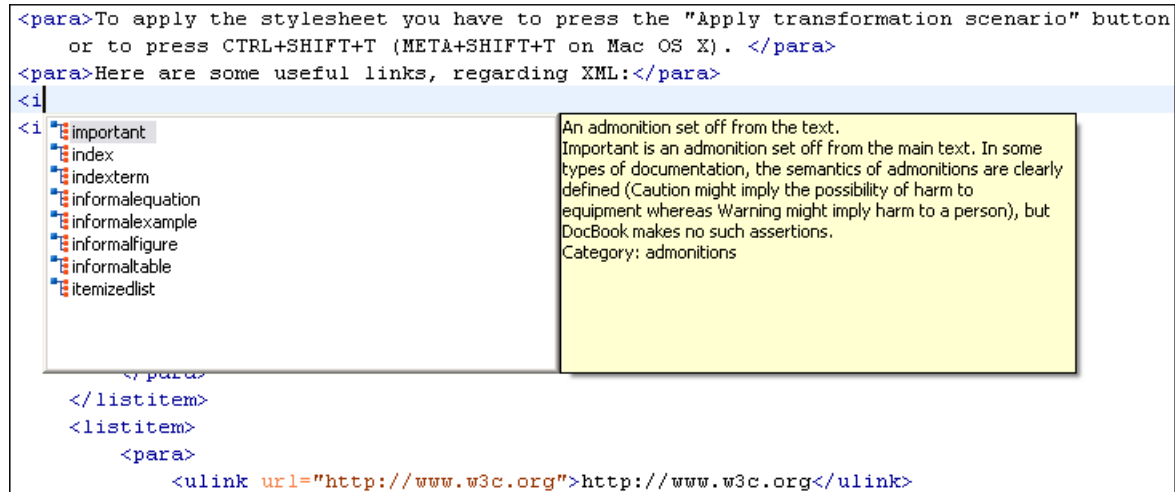



Figure 68: Content Completion Driven by DocBook DTD

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.

Go to menu **Document > Schema > Associate schema...** or click the  **Associate schema** toolbar button to select the schema that will be associated with the XML document. The following dialog is displayed:

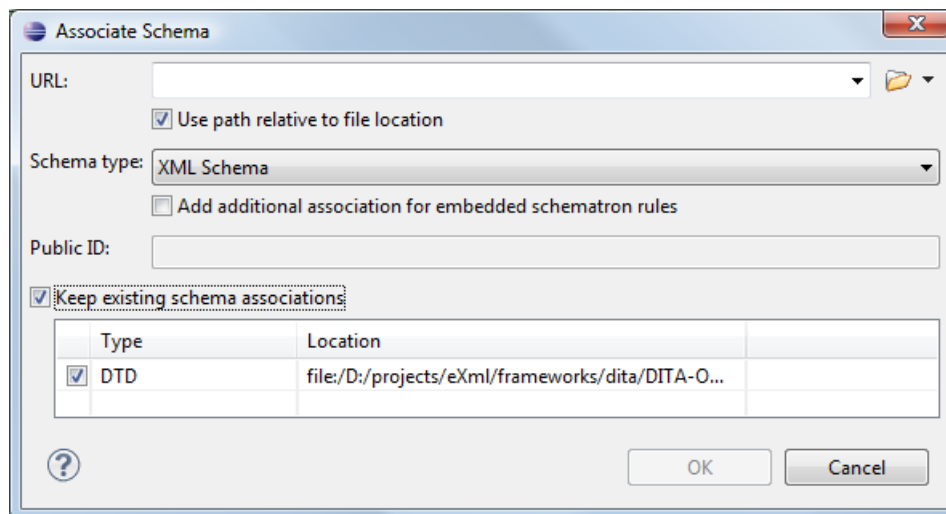


Figure 69: The Associate Schema Dialog

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);
- **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 has the following attributes:

- `href` - schema file location. Mandatory attribute.
- `type` - content type of schema. Optional attribute with the following possible values:
 - for DTD the recommended value is `application/xml-dtd`;
 - for W3C XML Schema the recommended value is `application/xml` or can be left unspecified;
 - for RELAX NG the recommended value is `application/xml` or can be left unspecified;
 - for RELAX NG - compact syntax the recommended value is `application/relax-ng-compact-syntax`;
 - for Schematron the recommended value is `application/xml` or can be left unspecified;
 - for NVDL the recommended value is `application/xml` or can be left unspecified.
- `schematypens` - namespace of schema language of referenced schema with the following possible values:
 - for DTD - not specified;
 - for W3C XML Schema the recommended value is `http://www.w3.org/2001/XMLSchema`;

- for RELAX NG the recommended value is `http://relaxng.org/ns/structure/1.0`;
- for RELAX NG - not specified;
- for Schematron the recommended value is `http://purl.oclc.org/dsdl/schematron`;
- for NVDL the recommended value is `http://purl.oclc.org/dsdl/nvd1/ns/structure/1.0`.
- `phase` - phase name of validation function in Schematron schema. Optional attribute.
- `title` - title for associated schema. Optional attribute.

Older versions of Oxygen XML Editor plugin used the `oxygen` processing instruction with the following attributes:

- `RNGSchema` - specifies the path to the Relax NG schema associated with the current document;
- `type` - specifies the type of Relax NG schema. It is used together with the `RNGSchema` attribute and can have the value "xml" or "compact";
- `NVDLSchema` - specifies the path to the NVDL schema associated with the current document;
- `SCHSchema` - specifies the path to the SCH schema associated with the current document.



Note: Documents that use the `oxygen` processing instruction are compatible with newer versions of Oxygen XML Editor plugin.

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 plugin 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 plugin 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 plugin 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 **XML > Learn Structure > Ctrl (Meta on Mac OS) + Shift + L**.
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 **XML > Save Structure > Ctrl (Meta on Mac OS) + Shift + S**. Enter the DTD file path.
4. Press the *Save* button.

Streamline with Content Completion

The intelligent **Content Completion Assistant** available in Oxygen XML Editor plugin enables rapid, in-line identification and insertion of structured language elements, attributes and, in some cases, their parameter options.

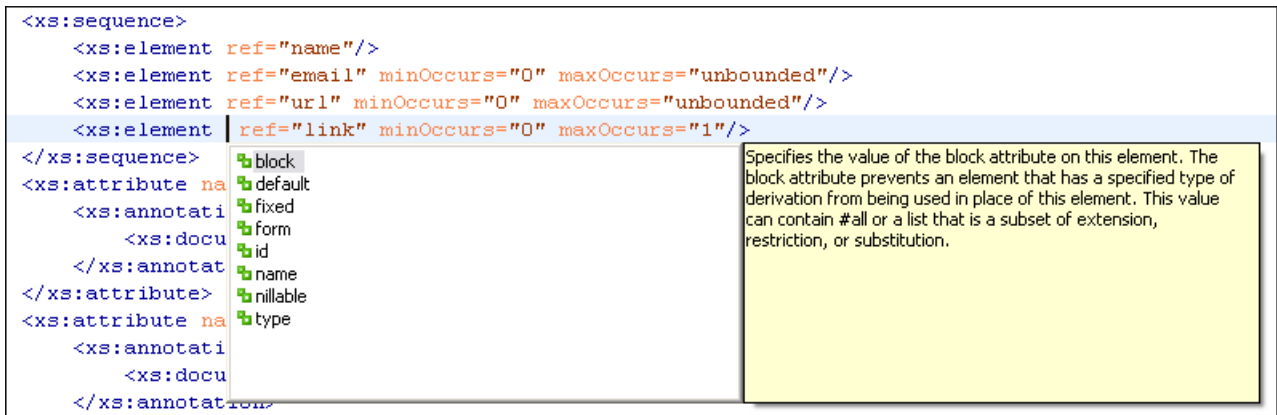


Figure 70: Content Completion Assistant

The functioning of the **Content Completion Assistant** feature is schema-driven (XML Schema, DTD, and RELAX NG). When Oxygen XML Editor plugin detects a schema, it logs its URL in the *Status view*.

The **Content Completion Assistant** is enabled by default, but you can disable it from *Window > Preferences > Oxygen XML Editor > 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 (Meta on Mac OS)+Space on a partial element or attribute name.



Note: If the Content Completion list contains only one valid proposal, when you press the CTRL (Meta on Mac OS)+Space shortcut key, the proposal is automatically inserted.

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 (Meta on Mac OS)+ Enter on your keyboard. The application 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 **Preferences > Editor > 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, select the **Add optional content** and **Add first Choice particle** check boxes in the **Preferences > Editor > Content Completion** preferences page.

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.

If the **Content Completion Assistant** proposals list contains only one element, the list is not displayed. When you trigger the **Content Completion Assistant**, the element is inserted automatically at the caret position.

A schema may declare certain attributes as *ID* or *IDREF/IDREFS*. When the document is validated, Oxygen XML Editor plugin 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 plugin 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 plugin [Content Completion options](#). Oxygen XML Editor plugin 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 plugin [Content Completion options](#). Oxygen XML Editor plugin 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:

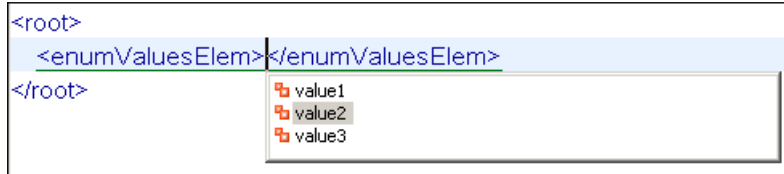


Figure 71: Content Completion assistant - element values in Relax NG documents

Schema Annotations

If the document's schema is an XML Schema, Relax NG (full syntax), NVDL or DTD and it contains element, attributes or attributes values annotations, these will be presented when the content completion window is displayed, only if the option *Show annotations* is enabled. Also the annotation is presented in a small tooltip window displayed automatically when the mouse hovers over an element or attribute annotated in the associated schema of the edited document.

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>

```

If the current element / attribute in the edited document does not have an annotation in the schema and that schema is an XML Schema, Oxygen XML Editor plugin seeks an annotation in the type definition of the element / attribute or, if no annotation is found there, in the parent type definition of that definition, etc.

When editing a Schematron schema 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. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.5.0.1 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion displays also the XSLT Saxon extension functions as in the following figure:

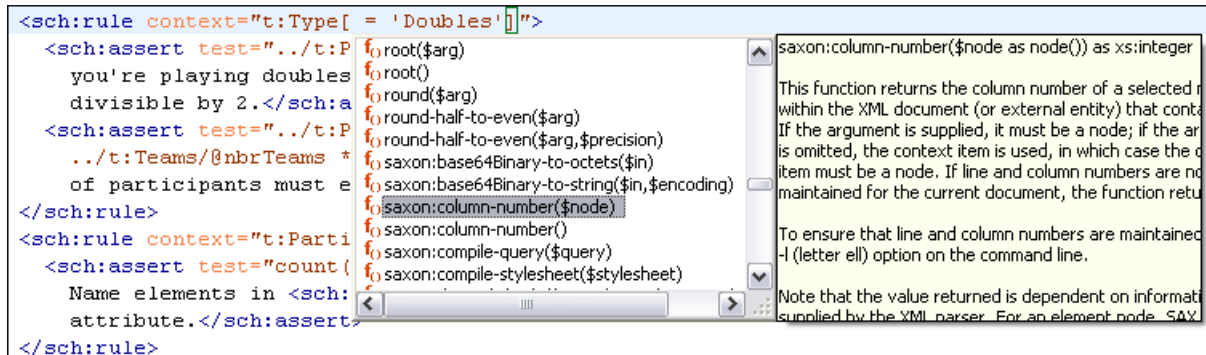


Figure 72: XSLT extension functions in Schematron schemas documents

In a Relax NG schema 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 together with the content completion window:

For NVDL schemas annotations for the elements / attributes in the referred schemas (XML Schema, RNG, etc) are presented

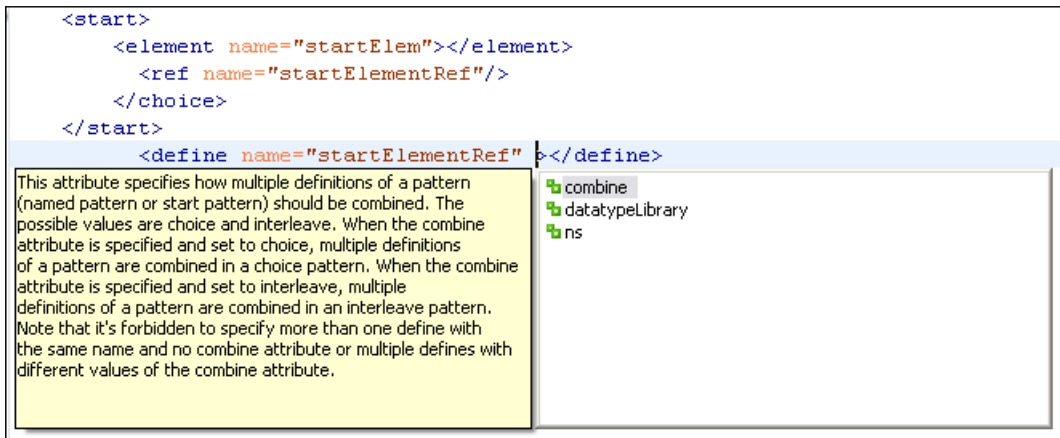


Figure 73: Schema annotations displayed at Content Completion

The following HTML tags are recognized inside the text content of an XML Schema annotation: `p`, `br`, `ul`, `li`. They are rendered as in 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.

For DTD Oxygen XML Editor plugin defines a custom mechanism for annotation using comments enabled from the option [Use DTD comments as annotations](#). The text of a comment with the following format will be presented on content completion:

```
<!--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 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. Open the **Model** view from **Window > Show View > Other > oXygen XML Editor > Model view**

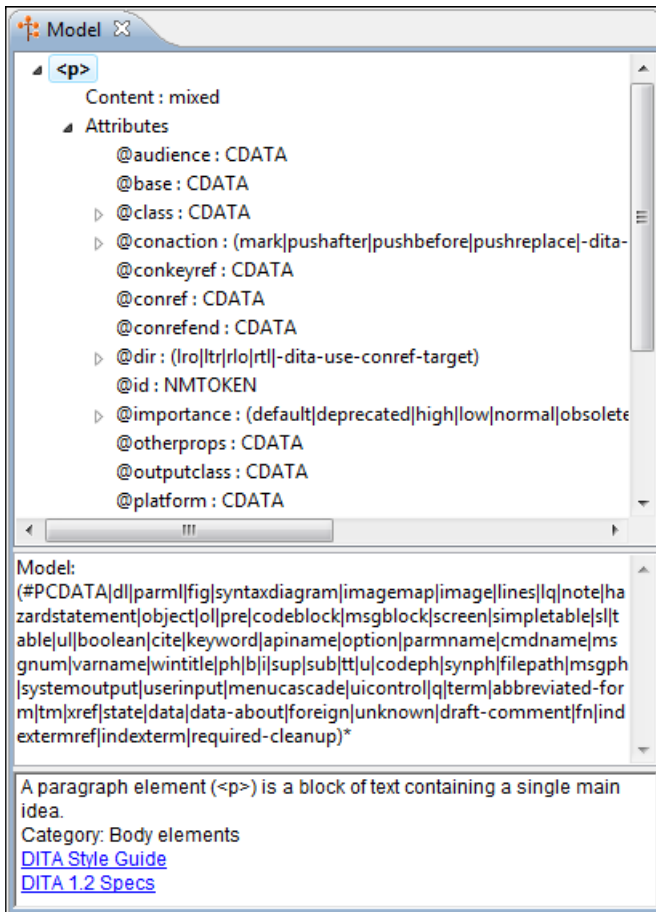


Figure 74: The Model View

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.

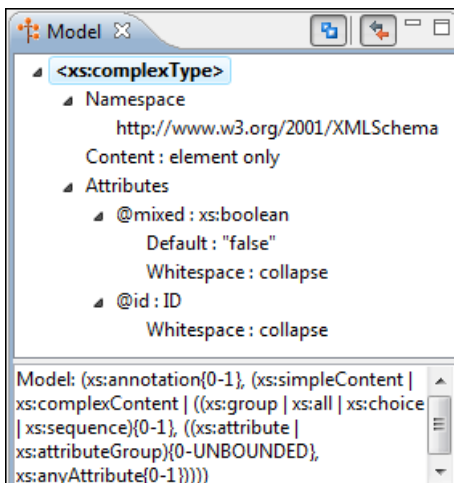


Figure 75: The Element Structure Panel

The Annotation Panel

The **Annotation** panel displays the annotation information for the currently selected element. This information is collected from the XML schema.

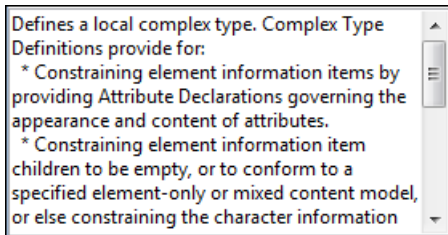


Figure 76: The Annotation panel

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.

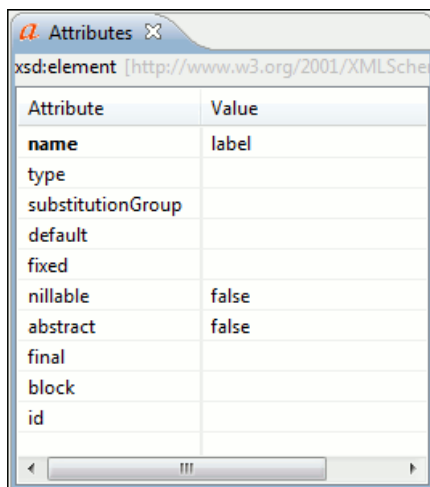


Figure 77: The Attributes View

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 document's schema. 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.

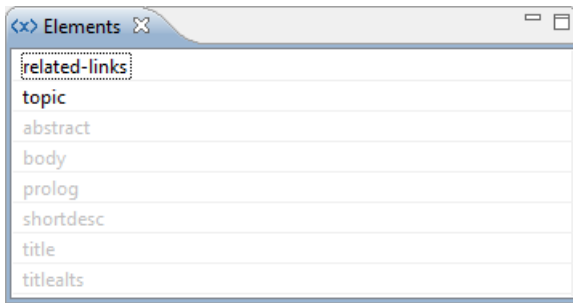


Figure 78: The Elements View

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.

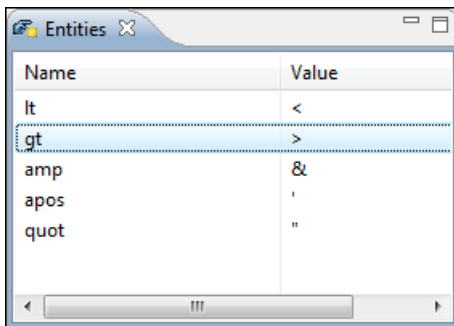


Figure 79: The Entities View

Code Templates

You can define short names for predefined blocks of code called *code templates*. The short names are displayed in the Content Completion window when the word at cursor position is a prefix of such a short name. If there is no prefix at cursor position (a whitespace precedes the cursor), all the code templates are listed.

Oxygen XML Editor plugin comes with numerous predefined code templates. You can also *define* your own code templates for any type of editor. For more details, see the [example for XSLT editor code templates](#).

To obtain the template list, you use the **Ctrl (Meta on Mac OS) + Space** content completion shortcut key, or the **Ctrl (Meta on Mac OS) + Shift + Space** code templates shortcut key. The first shortcut displays the code templates in the same *content completion list with elements from the schema of the document*. The second shortcut displays only the code templates and is the default shortcut of the action **Document > Content Completion > Show Code Templates**.

The syntax of the code templates allows you to use the following *editor variables*:

- **#{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. Can have one of the following values:
 - url - input is considered an URL. Oxygen XML Editor plugin checks that the URL is valid before passing it to the transformation;
 - password - input characters are hidden;

- `generic` - the input is treated as generic text that requires no special handling;
- `relative_url` - input is considered an URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing;



Note: You can use the `$ask` editor variable in file templates. In this case, Oxygen XML Editor plugin keeps an absolute URL.

- `combobox` - displays a dialog that contains a non-editable combo-box;
- `editable_combobox` - displays a dialog that contains an editable combo-box;
- `radio` - displays a dialog that contains radio buttons;
- `'default-value'` - optional parameter. Provides a default value in the input text box;

Examples:

- `ask('message')` - Only the message displayed for the user is specified.
 - `ask('message', generic, 'default')` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
 - `ask('message', password)` - 'message' is displayed, the characters typed are masked with a circle symbol.
 - `ask('message', password, 'default')` - same as before, the default value is 'default'.
 - `ask('message', url)` - 'message' is displayed, the parameter type is URL.
 - `ask('message', url, 'default')` - same as before, the default value is 'default'.
- **`\${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;
 - **`\${cfd}** - 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 `frameworks` subfolder of the Oxygen XML Editor plugin installation folder;
 - **`\${pd}** - Current project folder as file path. Usually the current folder selected in the Project View;
 - **`\${oxygenInstallDir}** - Oxygen XML Editor plugin 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>.

To watch our video demonstration about code templates, go to http://oxygenxml.com/demo/Code_Templates.html.

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 plugin 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*, go to **Document > Validate > Check Well-Formedness**

(**Alt+Shift+V W (Cmd+Alt+V W on Mac)**). You can also open the drop-down menu of the  validate button on the toolbar and select  **Check Well-Formedness**. 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 files contained in the current project and selected with the mouse in *the Project view* can be checked for well-formedness with one action available on the popup menu of the Project view in the **Validate** submenu:  **Check Well-Formedness**.

Validating XML Documents Against a Schema

A *Valid* XML document is a *Well Formed* XML document, which also conforms to the rules of a schema which 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.
- Middle area where the error markers are depicted in red . The number of markers shown can be limited by modifying the setting **Window > Preferences > Oxygen XML Editor > 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.

Status messages from every validation action are logged into the *Console view*.

If you want to see all the validation error messages *grouped in a view* you should run the action **Validate** which is available both on the toolbar and on the **XML** menu. This action collects all error messages in the **Problems** view of the Eclipse platform if the validated file is in the current workspace or in a custom Oxygen view called **Errors** if the validated file is outside the workspace.

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 plugin *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*.

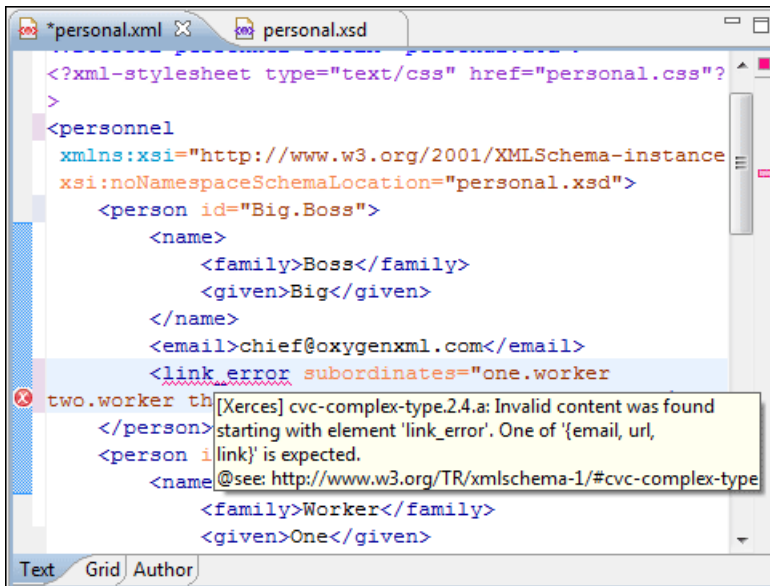


Figure 80: Automatic Validation on the Edited Document

Custom Validators

If you need to validate the edited document with other validation engine than the built-in one you have the possibility to configure external validators in the Oxygen XML Editor plugin user preferences. After such a custom validator is *properly configured* it can be applied on the current document with just one click on the **Custom Validation Engines** toolbar. The document is validated against the schema declared in the document.

Some validators are configured by default but they are third party processors which do not support the *output message format* of Oxygen XML Editor plugin for linked messages:

- **LIBXML** - Included in Oxygen XML Editor plugin (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 (`--xininclude`) 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 plugin 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 Mac 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 like:

```
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 plugin. 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 plugin (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 plugin (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 plugin. 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 plugin* for the [Oxygen-install-folder]/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 plugin. 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-install-folder]/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.

Linked Output Messages of an External Engine

Validation engines display messages in an output view at the bottom of the Oxygen XML Editor plugin 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 so that a click on the message in the output view highlights the location of the message in an editor panel containing the file referred in the message. This behavior is similar to the linked messages generated by the default built-in validator. The format for linked messages is:

- *Type*: [F|E|W] (the string *Type*: followed by a letter for the type of the message: fatal error, error, warning) - this line 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 line is optional in a linked message.
- *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
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 plugin 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 referring to parameters and templates 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 plugin (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 plugin 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 plugin, go to http://oxygenxml.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, go to **XML**. You can also open this dialog from the toolbar of the Oxygen XML Editor plugin.

The following dialog 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 plugin 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.

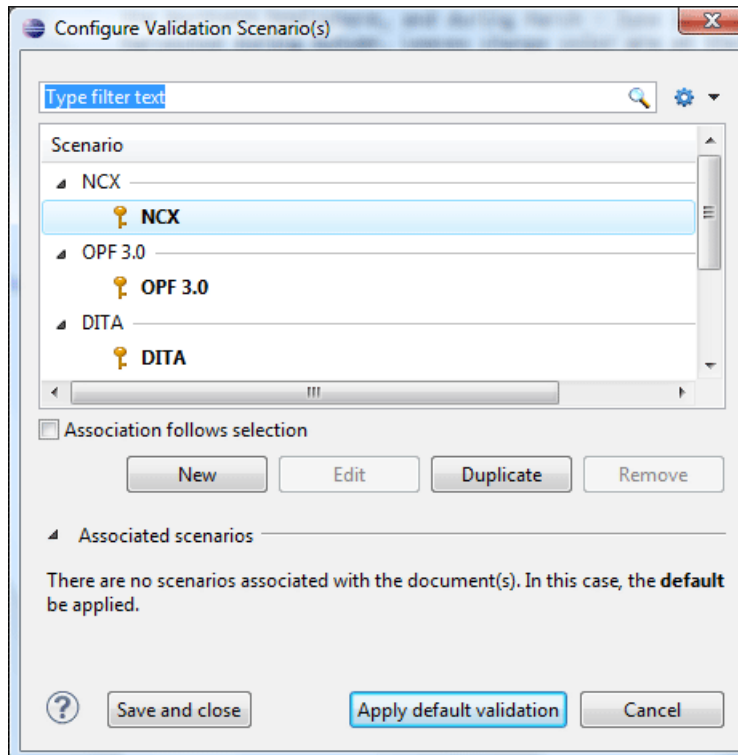


Figure 81: Configure Validation Scenario

2. Press the **New** button to add a new scenario.
3. Press the **Add** button to add a new validation unit with default settings. The dialog that lists all validation units of the scenario is opened.

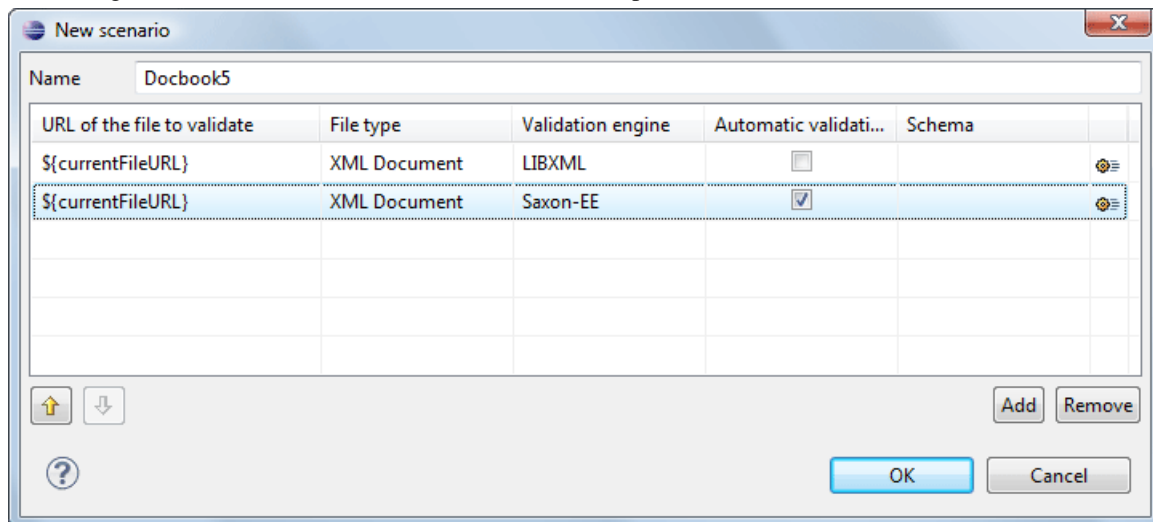


Figure 82: Add / Edit a Validation Unit


The table holds the following information:

- **Storage** - allows you to create a scenario at project level, or as global;
- **URL of the file to validate** - the URL of the main module which 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 validated in the current validation unit. Oxygen XML Editor plugin 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 plugin for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the default engine executes the validation. This engine is set in **Preferences** pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, and others) instead of a validation scenario;
- **Automatic validation** - if this option is checked, then the validation operation defined by this row of the table is applied also by *the automatic validation feature*. If the **Automatic validation** feature is *disabled in Preferences* then this option does not take effect as the Preference setting has higher priority;
- **Schema** - the this option becomes active when you set the **File type** to **XML Document**;
- **Settings** - opens the **Specify Schema** dialog box, allowing 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. Edit the URL of the main validation module.

Specify the URL of the main module:

- browsing for a local, remote, or archived file;
- using an *editor variable* or a *custom editor variable*, available in the following pop-up menu, opened after pressing the  button:

```

${start-dir} - Start directory of custom validator
${standard-params} - List of standard parameters
${cfn} - The current file name without extension
${currentFileURL} - The path of the currently edited file (URL)
${cfdu} - The path of current file directory (URL)
${frameworks} - Oxygen frameworks directory (URL)
${pdu} - Project directory (URL)
${oxygenHome} - Oxygen installation directory (URL)
${home} - The path to user home directory (URL)
${pn} - Project name
${env(VAR_NAME)} - Value of environment variable VAR_NAME
${system(var.name)} - Value of system variable var.name

```

Figure 83: Insert an Editor Variable

5. Select the type of the validated document.

Note that it 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 *automatic validation feature is turned on in Preferences*.

8. Choose what schema is used during validation: the one detected after parsing the document or a custom one.


Validation Actions in the User Interface

To validate the currently edited document, use one of the following methods:


- Go to **XML > Validate Document (Alt+Shift+V V) ((Cmd+Alt+V V on Mac OS))** or click the  **Validate** button from the **Validate** toolbar. An error list is presented in the message panel. Mark-up 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.
- Go to **XML > Validate (cached)** or click the  **Validate (cached)** button from the **Validate** toolbar. This action caches the schema, allowing it to be reused for the next validation. Mark-up 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.

- Go to **XML > Validate with (Alt+Shift+V E) ((Cmd+Alt+V E on Mac OS))** or click the  **Validate with** button from the **Validate** toolbar. 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. Mark-up of current document is checked to conform with the specified schema rules.

- Select submenu **Batch Validation > Validate** in the contextual menu of **Navigator** or **Package Explorer** view, to validate all selected files with their declared schemas.
- Select **Batch Validation > Validate With ...** from the contextual menu of the **Navigator** or **Package Explorer** view, to choose a schema and validate all selected files with it.
- Go to **XML > Clear Validation Markers (Alt+Shift+V X) ((Cmd+Alt+V X on Mac OS))** or click the  **Clear Validation Markers** button to clear the error markers added to the **Problems** view at the last validation of the current edited document.
- Select the submenu **Batch Validation > Configure Validation Scenario ...** of the contextual menu of **Navigator** or **Package Explorer** view, to configure and apply a validation scenario in one action to all the selected files in the **Navigator** or **Package Explorer** view.

Also you can select several files in the views **Package Explorer** or **Navigator** and validate them with one click by selecting the action **Validate selection**, the action **Validate selection with Schema ...** or the action **Configure Validation Scenario ...** available from the contextual menu of that view, the submenu **Batch Validate**.

In case too many validation errors are detected and the validation process takes too long, you can [limit the maximum number of reported errors from the Preferences page](#).

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.






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.

Figure 84: Folding of the XML Elements

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 (Ctrl+Alt+Y)** available from the contextual menu

Other menu actions related to folding of XML elements are available from the contextual menu of the current editor:

- **Ctrl (Meta on Mac OS) + NumPad +/- > Document > Folding >  Close Other Folds > Ctrl (Meta on Mac OS) + NumPad +/-** - Folds all the elements except the current element.
- **Document > Folding >  Collapse Child Folds (Ctrl+Decimal) (Ctrl+NumPad+-) ((Cmd+NumPad+- on Mac OS))** - Folds the elements indented with one level inside the current element.
- **Document > Folding >  Expand Child Folds (Ctrl+NumPad++) ((Cmd+NumPad++))** - Unfolds all child elements of the currently selected element.
- **Document > Folding >  Expand All (Ctrl+NumPad+*) ((Cmd+NumPad+* on Mac OS))** - Unfolds all elements in the current document.
- **Document > Folding >  Toggle Fold (Alt+Shift+Y) ((Cmd+Alt+Y on Mac OS))** - Toggles the state of the current fold.

You can use folding by clicking on the special marks displayed in the left part of the document editor.

To watch our video demonstration about the folding support in Oxygen XML Editor plugin, go to <http://oxygenxml.com/demo/FoldingSupport.html>.

Outline View

The Outline view offers the following functionality:

- *XML Document Overview* on page 146
- *Outline Specific Actions* on page 146
- *Modification Follow-up* on page 147
- *Document Structure Change* on page 147
- *Document Tag Selection* on page 147

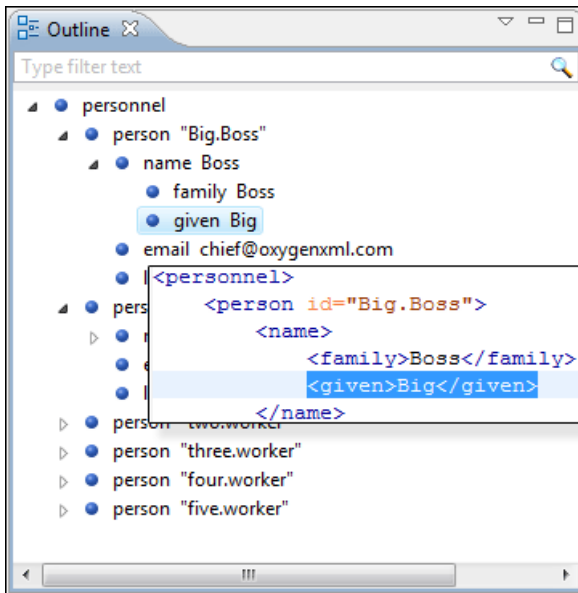


Figure 85: The Outline View

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 (Meta on Mac OS) + Click**.



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 **View** 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 (Meta on Mac OS))** 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 Popup Menu of the Outline Tree

The *Append Child*, *Insert Before* and *Insert After* submenus of the outline tree popup menu allow to quickly insert new tags in the document at the place of the element currently 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 *Insert Before* and *Insert After* submenus of the Outline tree popup menu 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.

Edit attributes for the selected node. A dialog is presented allowing the user to see and edit the attributes of the selected node.

The *Toggle comment* item of the outline tree popup menu is the same item as in the editor popup menu with the same name. It encloses the currently selected element of the outline tree in an XML comment, if the element is not commented, or uncomments it, if it is commented.

The *Cut*, *Copy* 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 outline tree (Cut, Copy, Paste).

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 also use key search to look for a particular tag name in the Outline tree.

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 referred;
- 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 referred 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 Decl.). 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 the DOCTYPE Decl. 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 plugin 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 % xinclude SYSTEM "../frameworks/docbook/dtd/xinclude.mod">
%include;
]>
<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 Decl. 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 plugin is turned on by default. You can *toggle it* by going to the entry **Enable XInclude processing** in the menu **Window > Preferences ... > oXygen XML Editor > XML > XML Parser**.

When enabled, Oxygen XML Editor plugin 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 referred 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 plugin 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 new catalog elements *systemSuffix* and *uriSuffix*.

Depending on the resource type, Oxygen XML Editor plugin uses different catalog mappings.

Table 5: Catalog Mappings

Document	Referred Resource	Mappings
XML	DTD	<p><i>system</i> or <i>public</i></p> <p>The <i>Prefer</i> option controls which one of the mappings should be used.</p>
	XML Schema	<p>The following strategy is used (if one step fails to provide a resource, the next is applied):</p> <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	<p>This happens only if the <i>Resolve schema locations also through system mappings</i> option is enabled (it is by default);</p>
	NVDL	<ol style="list-style-type: none"> 3. resolve the root <i>namespace</i> using <i>uri</i> catalog mappings.
XSL	XSL/ANY	<i>uri</i>
CSS	CSS	<i>uri</i>
XML Schema	XML Schema	<p>The following strategy is used (if one step fails to provide a resource, the next is applied):</p> <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.
Relax NG	Relax NG	<p>This happens only if the <i>Resolve schema locations also through system mappings</i> option is enabled (it is by default);</p> <ol style="list-style-type: none"> 3. resolve schema <i>namespace</i> using <i>uri</i> catalog mappings. <p>This happens only if the <i>Process namespaces through URI mappings for XML Schema</i> option is enabled (it is not by default);</p>

An XML Catalog file can be created quickly in Oxygen XML Editor plugin 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 plugin comes with a built-in catalog set as default, but you can also create your own one. Oxygen XML Editor plugin 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 plugin resolves the location of a schema in the following order:

- First, the schema location is attempted to be resolved as an URI (`uri`, `uriSuffix`, `rewriteURI` from the XML catalog). If this succeeds, the process end here.
- In case the **Resolve schema locations also through system mappings** option is selected, the schema location is attempted to be resolved 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, the namespace of the schema is attempted to be resolved as an URI (`uri`, `uriSuffix`, `rewriteURI` from the XML catalog). If this succeeds, the process ends here.
- If none of the previous attempts succeeded, 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 *XIinclude* 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 > Other > oXygen > Resource Hierarchy/Dependencies**. As an alternative, right click the current document and either select **Resource Hierarchy** or **Resource Dependencies**.

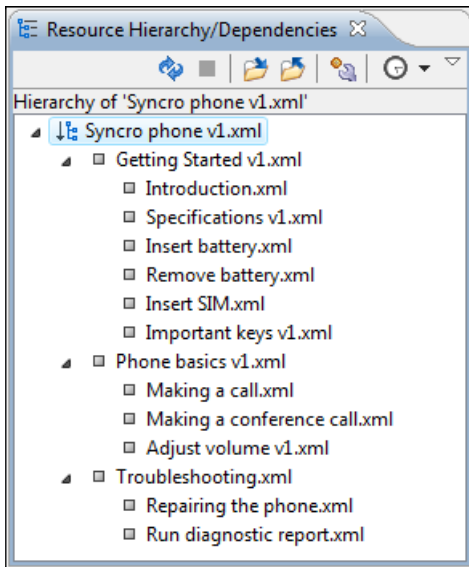


Figure 86: Resource Hierarchy/Dependencies View - Hierarchy for `Syncro phone v1.xml`

The build process for the dependencies view is started with the **Resource Dependencies** action available on the contextual menu.

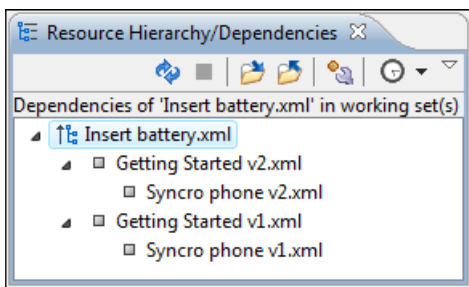









Figure 87: Resource Hierarchy/Dependencies View - Dependencies for `Insert battery.xml`

The following actions are available in the **Resource Hierarchy/Dependencies** view:

-  - Refreshes the Hierarchy/Dependencies structure;
-  - Stop the hierarchy/dependencies computing;
-  - Allows you to choose a resource to compute the hierarchy structure;
-  - Allows you to choose a resource to compute the dependencies structure;
-  - 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;
-  - Repeats a previous dependencies computation.

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 *XIinclude* 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 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 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 is available for both actions. The **Preview** dialog presents a list with the resources that are updated.

Converting Between Schema Languages

The **Generate/Convert Schema** 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 plugin will generate an approximation of the source schema.

The conversion functionality is available from **XML Tools > Generate/Convert Schema...** (**Ctrl + Shift + "/"** (**Meta + Shift + /** on Mac OS)) and from the toolbar button  **Convert to...** .

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.

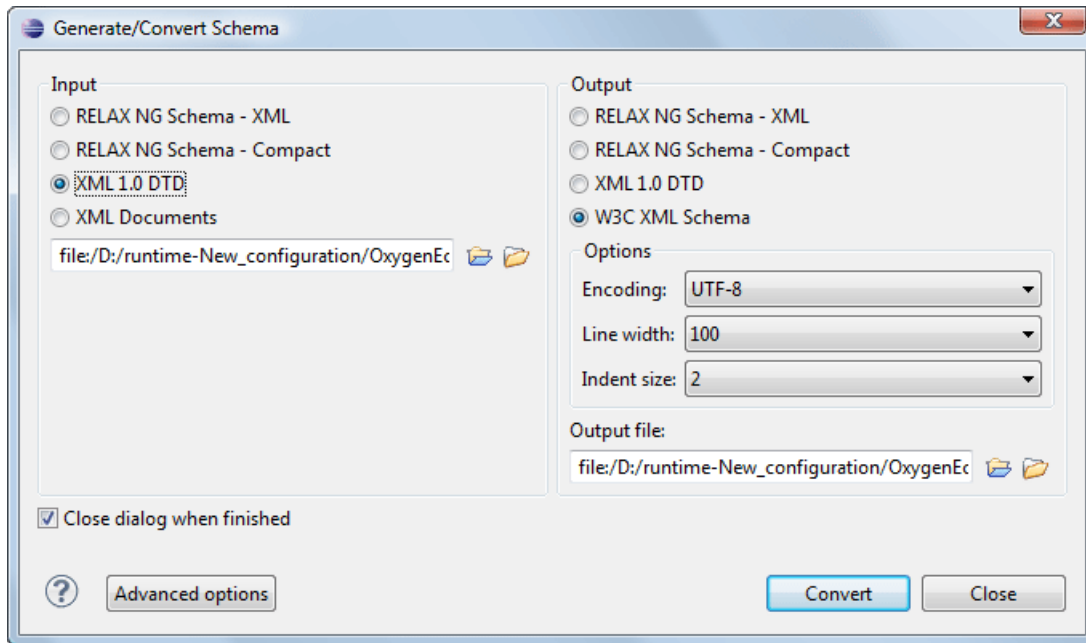


Figure 88: Convert a Schema to Other Schema Language

The language of the target schema is specified with one of the four radio buttons of the **Output** panel. The encoding, the maximum line width and the number of spaces for one level of indentation can be also specified in this panel.

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 advanced options are:

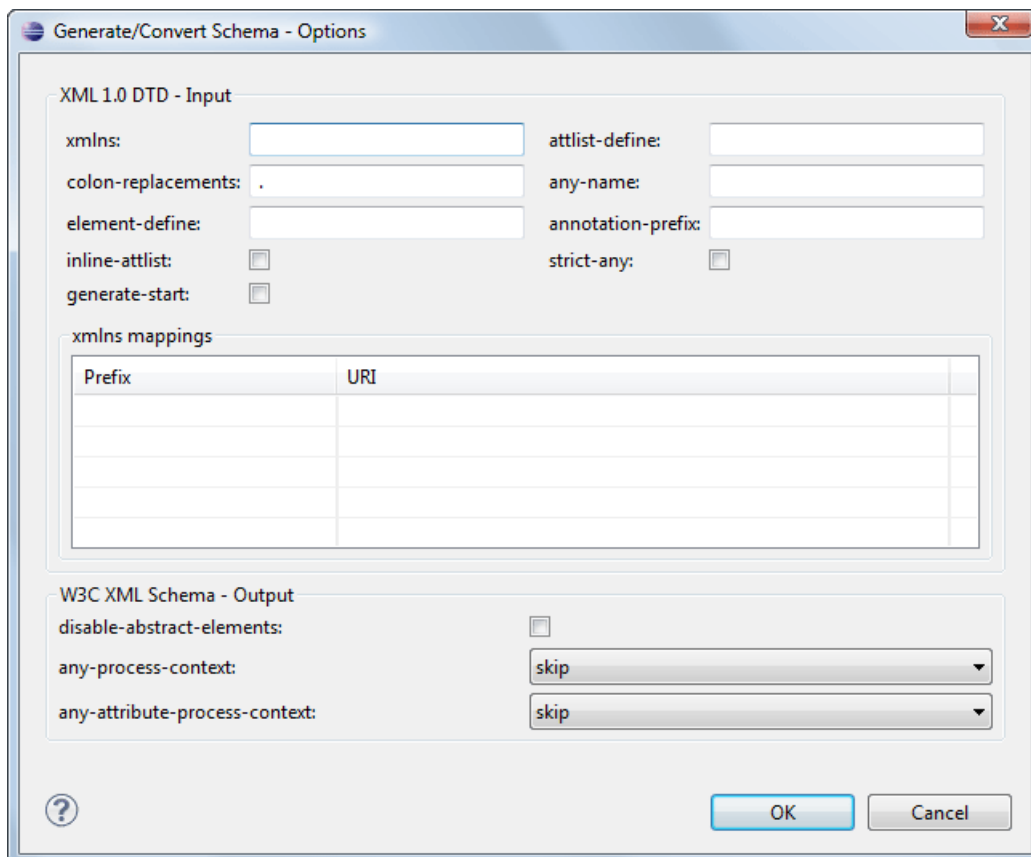


Figure 89: Convert a Schema to Other Schema Language - Advanced Options

For the **Input** panel:

- `xmlns` field - Specifies the default namespace, that is the namespace used for unqualified element names;
- `xmlns` table - Each row specifies in 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-attlist` - 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;
- `attlist-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, Trang uses a wildcard that allows any element;
- `generate-start` - Specifies whether Trang should generate a start element. DTD's do not indicate what elements are allowed as document elements. Trang assumes that all elements that are defined but never referenced are allowed as document elements;
- `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, Trang will use a for prefix unless that conflicts with a prefix used in the DTD.

For the **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).

Formatting and Indenting Documents (Pretty Print)

In structured markup languages, the whitespace between elements that is created using the *Space bar*, *Tab* or multiple line breaks is not recognized by the parsing tools. Often this means that when structured markup documents are opened, they are arranged as one long, unbroken line, that seems to be a single paragraph.

While this is a perfectly acceptable practice, it makes editing difficult and increases the likelihood of errors being introduced. It also makes the identification of exact error positions difficult. Formatting and Indenting, also called **Pretty Print**, enables such documents to be neatly arranged, in a manner that is consistent and promotes easier reading on screen and in print output.

Pretty print is in no way associated with the layout or formatting that will be used in the transformed document. This layout and formatting is supplied by the XSL stylesheet specified at the time of transformation.

To change the indenting of the current selected text see the [Indent selection](#) action.

For user preferences related to formatting and indenting like **Detect indent on open** and **Indent on paste** see [the corresponding Preferences panel](#).

XML elements can be excepted from the reformatting performed by the pretty-print operation by including them in the *Preserve space elements (XPath)* list. That means that when the *Format and Indent* (pretty-print) action encounters in

the document an element with the name contained in this list, the whitespace is preserved inside that element. This is useful when most of the elements must be reformatted with the exception of a few ones which are listed here.

For the situation when whitespace should be preserved in most elements with the exception of a few elements, the names of these elements must be added to the *Strip space elements (XPath)* list.

In addition to simple element names, both the *Preserve space elements (XPath)* list and the *Strip space elements (XPath)* one accept a restricted set of XPath expressions to cover a pattern of XML elements with only one expression. The allowed types of expressions are:

//xs:documentation	the XPath descendant axis can be used only at the beginning of the expression; the namespace prefix can be attached to any namespace, no namespace binding check is performed when applying the pretty-print operation
/chapter/abstract/title	note the use of the XPath child axis
//section/title	the descendant axis can be followed by the child axis

The value of an *xml:space* attribute present in the XML document on which the pretty-print operation is applied always takes precedence over the *Preserve space elements (XPath)* and the *Strip space elements (XPath)* lists.

How to use zero size indent

When you use pretty print, or when you save a document from the **Author**, Oxygen XML Editor plugin allows you to use zero size indent. To stop indenting text depending on the depth of the nodes in an XML document:

1. Go to **Option > Preferences > Editor > Format**;
2. Disable **Detect indent on option**;
3. Set **Indent size** to zero.

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 plugin 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 Navigator 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 plugin 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 plugin 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 refactory actions* for ID/IDREFS values and for updating references when renaming/moving a resource. Oxygen XML Editor plugin performs the search and refactory 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 plugin 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, go to *Options Preferences Editor Mark Occurrences*.

Search and Refactor Actions for ID/IDREFS

Oxygen XML Editor plugin 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 and 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 (Meta on Mac OS) + Click** 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 plugin 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.

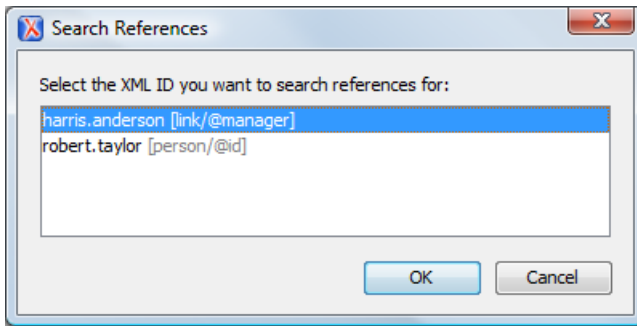


Figure 90: Selecting an ID in the Author Mode

Quick Assist Support for ID/IDREFS in Text Mode

The Quick Assist support is activated automatically when you place the caret inside an 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 **Ctrl + 1** (Meta 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;
- **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


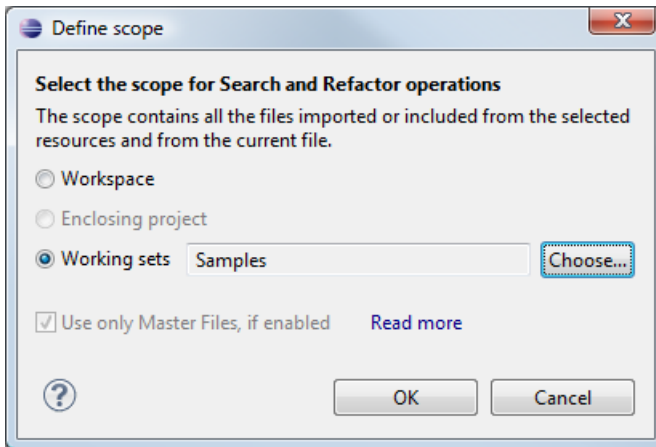
The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the  **Change scope** operation, available in the Quick Fix action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project 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 91: 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 **Console** view allowing you to monitor how the operation is being executed.

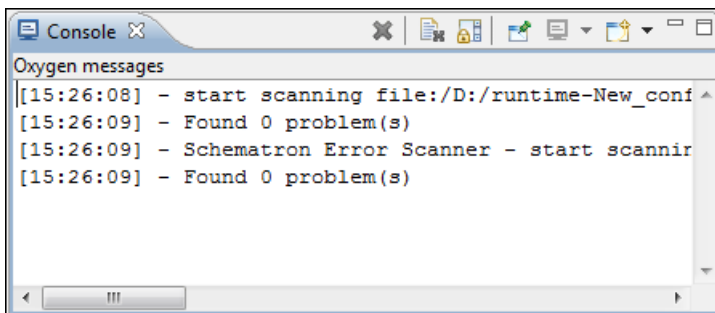


Figure 92: The Console view messages

Messages contain a timestamp, the name of the thread that generated it and the actual status information. The number of displayed messages in the **Console** view can be controlled from the [Options panel](#).

In order to make the view visible go to menu **Window > Show View > Console**.

XML Editor Specific Actions

Oxygen XML Editor plugin offers groups of actions for working on single XML elements. They are available from the the context menu of the main editor panel.

Edit Actions

The following XML specific editing actions are available in Text mode:

- **contextual menu of current editor > Toggle comment (Ctrl (Meta on Mac OS) + /)** - 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.






Select Actions

In Text mode of the XML editor these actions are enabled when the caret is positioned inside a tag name:

- **contextual menu of current editor > Select > Element** - Selects the entire current element;
- **contextual menu of current editor > 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;
- **contextual menu of current editor > Select > Attributes** - Selects all the attributes of the current element;
- **contextual menu of current editor > Select > Parent** - Selects the parent element of the current element;
- Double click an element or processing instruction - If the double 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 double 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;
- Double click an attribute in **Text** mode - If the double click is performed before the start tag of an attribute or after its end tag, the entire attribute is selected by the double 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 can be applied on the text content of the XML editor:

- **contextual menu of current editor > Source >  Escape Selection ...** - Escapes a range of characters by replacing them with the corresponding character entities.
- **contextual menu of current editor > Source >  Unescape Selection ...** - Replaces the character entities with the corresponding characters.
- **contextual menu of current editor > Source >  Indent selection (Ctrl + I) ((Cmd + I on Mac OS))** - Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences of the user*.
- **contextual menu of current editor > Source >  Format and Indent Element (Ctrl + Shift + I)** - Pretty prints the element that surrounds the caret position.
- **contextual menu of current editor > Source >  Insert XInclude** - Shows a dialog that allows you to browse and select the content to be included and generates automatically 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 only the **source** tab is presented.

- **contextual menu of current editor > Source >  Import entities list** - Shows 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, if choosing the file `chapter1.xml` and `chapter2.xml`, the following section is inserted in the DOCTYPE:

```
<!ENTITY chapter1 SYSTEM "chapter1.xml">
<!ENTITY chapter2 SYSTEM "chapter2.xml">
```

- **contextual menu of current editor > Join and normalize** - The action works on the selection. It 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.

XML Document Actions

The **Text** mode of the XML editor provides the following document level actions:

- **contextual menu of current editor > Show Definition** - 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 referred in the value of the attribute.



Note: Alternatively you can use any of the following shortcuts:

- **Ctrl (Meta on Mac OS) + Shift + ENTER** on your keyboard;
 - **Ctrl (Meta on Mac OS) + Click** an element or attribute name.
- **contextual menu of current editor > Copy XPath (Ctrl (Meta on Mac OS)+Shift+.)** - Copies the XPath expression of the current element or attribute from the current editor to the clipboard.
 - **contextual menu of current editor > Go To > Go to Matching Tag** - Moves the cursor to the end tag that matches the start tag, or vice versa.
 - **contextual menu of current editor > Go to > Go after Next Tag (Ctrl (Meta on Mac OS)+])** - Moves the cursor to the end of the next tag.
 - **contextual menu of current editor > Go to > Go after Previous Tag (Ctrl (Meta on Mac OS)+[)** - Moves the cursor to the end of the previous tag.
 - **XML > Associate XSLT/CSS Stylesheet** - Inserts an `xml-stylesheet` processing instruction at the beginning of the document referencing either an XSLT or a CSS file depending on the user selection. Either reference is useful for rendering the document in a Web browser when the action **Open in browser** is executed. 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 plugin fully implements the W3C recommendation regarding [Associating Style Sheets with XML documents](#). See also [Specifying external style sheets](#) in HTML documents.

You can use the **Ctrl (Meta on Mac OS) + Click** 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.

XML Refactoring Actions

The following refactoring actions are available while editing an XML document:




- **context menu of current editor > XML Refactoring > Surround with tag... (Alt+Shift+E) ((Cmd+Alt+E on Mac OS))** - 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. The caret is placed:
 - between the start and end tag, if the **Cursor position between tags** option is set;
 - at the end of the start tag, in an insert-attribute position, if the **Cursor position between tags** option is not set.
 - **context menu of current editor > XML Refactoring > Surround with <tag> (Alt+Shift+/) ((Cmd+Alt+/ on Mac OS))** - Similar in behavior with the **Surround with tag...** action, except that it inserts the last tag used by the **Surround with tag...** action.
 - **context menu of current editor > XML Refactoring > Rename element ((Cmd+Alt+R on Mac OS))** - 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.
- context menu of current editor > XML Refactoring > Rename prefix > Alt + Shift + P > ((Cmd+Alt+P on Mac OS))** - the prefix of the element from the caret position and the elements that have the same prefix as the current element can be renamed according with the options from the **Rename** dialog.

Selecting the **Rename current element prefix** option, the application will recursively traverse the current element and all its children.

For example, to change the `xmlns:p1="ns1"` association existing in the current element to `xmlns:p5="ns1"`, just select this option and press **OK**. If the association `xmlns:p1="ns1"` is applied on the parent of the current element, then Oxygen XML Editor plugin will introduce a new declaration `xmlns:p5="ns1"` in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated in the current element with another namespace, let's say `ns5`, then a dialog showing the conflict will be displayed. Pressing the **OK** button, the prefix will be modified from `p1` to `p5` without inserting a new declaration `xmlns:p5="ns1"`. On **Cancel** no modification is made.

Selecting the **Rename current prefix in all document** option, the application will apply the change on the entire document.

To apply the action also inside attribute values one must check the **Rename also attribute values that start with the same prefix** checkbox.

- **context menu of current editor > XML Refactoring >  Split element** - Split the element from the caret position in two identical elements. The caret must be inside the element.
- **context menu of current editor > XML Refactoring >  Join elements (Alt+Shift+F) ((Cmd+Alt+F on Mac OS))** - Joins the left and right elements relative to the current caret position. The elements must have the same name, attributes and attributes values.
- **context menu of current editor > XML Refactoring >  Delete element tags (Alt+Shift+,) ((Cmd+Alt+, on Mac OS))** - 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 plugin 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 pretty-print 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.

```

<xsl:template match="name">
  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end" font-weight="bold">Full Name:</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block text-align="start" color="red">
        <xsl:apply-templates select="*" />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>

```

Figure 93: Example of Coloring XML Tags by Prefix

Editor Highlights

An editor highlight is a text fragment emphasized by a colored background.

By default, Oxygen XML Editor plugin 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 from the [Options > Preferences > 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 which 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 plugin 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 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**.

Editing XHTML Documents

XHTML documents with embedded CSS, JS, PHP, and JSP scripts are rendered with dedicated coloring schemes. You can customize them in the **Window > Preferences > oXygen XML Editor > Editor > Syntax Highlight** preferences page.

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 plugin, go to http://oxygenxml.com/demo/XSL_Editing.html.

Validating XSLT Stylesheets

Oxygen XML Editor plugin 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 9.5.0.1 and *a JAXP transformer specified by the main Java class*. For XSLT 2.0, the options are: Saxon 9.5.0.1 and *a JAXP transformer specified by the main Java class*. For XSLT 3.0, the options are Saxon 9.5.0.1 and *a JAXP transformer specified by the main Java class*.


Custom Validation of XSLT Stylesheets

If you must validate an XSLT stylesheet with other validation engine than the Oxygen XML Editor plugin's built-in ones, you have the possibility to configure external engines as custom XSLT validation engines. After such a custom validator is *properly configured in Preferences* page, it can be applied on the current document with just one click on the **Custom Validation Engines** toolbar.

There are two validators configured by default:

- **MSXML 4.0** - included in Oxygen XML Editor plugin (Windows edition). It is associated to the XSL Editor type in *Preferences page*.
- **MSXML.NET** - included in Oxygen XML Editor plugin (Windows edition). It is associated to the XSL Editor type in *Preferences page*.

Associate a Validation Scenario

You are able to validate XSLT stylesheets using a validation scenario. To create a validation scenario, click  **Configure Validation Scenario(s)** in the main toolbar or go to the **XML > Configure Validation Scenario(s)**.

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 set in **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XSLT** is used. The list of reusable scenarios for documents of the same type as the current document is displayed in case you choose to use a custom validation scenario. 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 plugin 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 Navigator 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 plugin 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://oxygenxml.com/demo/MasterFilesSupport.html>.

Syntax Highlight

The XSL editor renders with dedicated coloring schemes the CSS and JS scripts, and XPath expressions. You can customize the coloring schemes in the **Window > Preferences > Oxygen XML Editor > Editor > Syntax Highlight** preferences page.

Content Completion in XSLT Stylesheets

The items in the list of proposals offered by the **Content Completion Assistant** is 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.

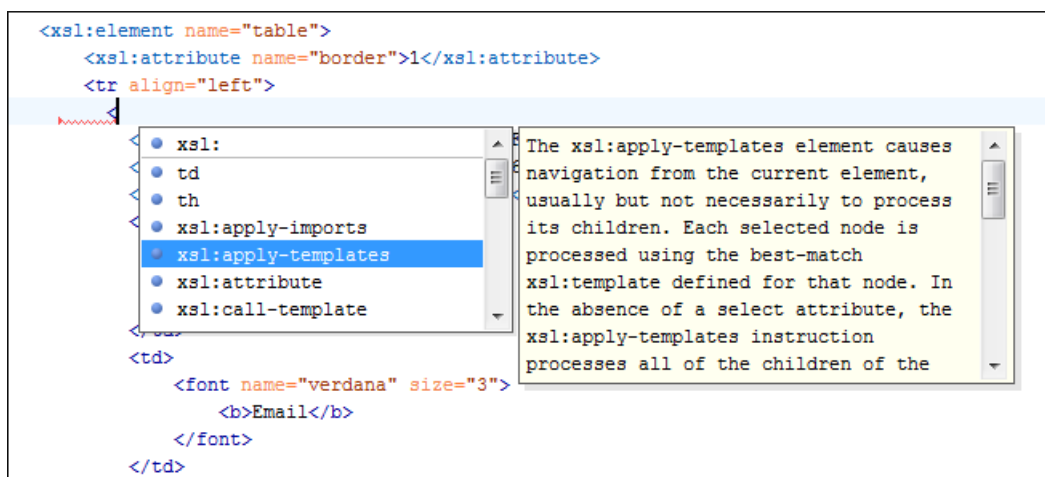


Figure 94: XSLT Content Completion Window

The **Content Completion Assistant** proposes the following item types, defined in the current stylesheet and in the imported and included XSLT stylesheets:

- template modes;
- template name;
- variable names;
- parameter names.



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 transformation engine 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 9.5.0.1 PE or Saxon 9.5.0.1 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 9.5.0.1 PE or Saxon 9.5.0.1 EE (for version 2.0 / 3.0);
- the validation engine specified in [Options](#) page is Saxon 6.5.5 (for version 1.0), Saxon 9.5.0.1 PE or Saxon 9.5.0.1 EE (for version 2.0 / 3.0).

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.

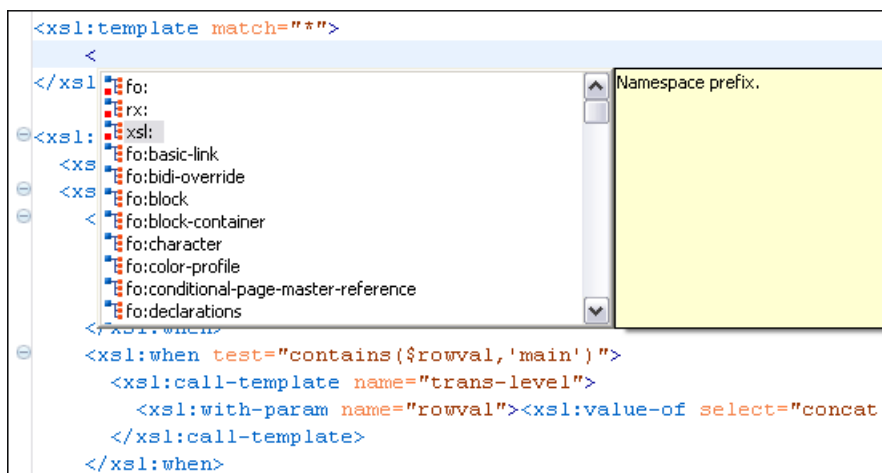


Figure 95: Namespace Prefixes in the Content Completion Window

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 plugin 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 (Meta on Mac OS) + Space** 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@oxygenxml.com</email>
    <link subordinates="one.worker"/>
  </person>
  <person id="one.worker">
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>
    <email>one@oxygenxml.com</email>
    <link manager="Big.Boss"/>
  </person>
</personnel>
```

and 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.

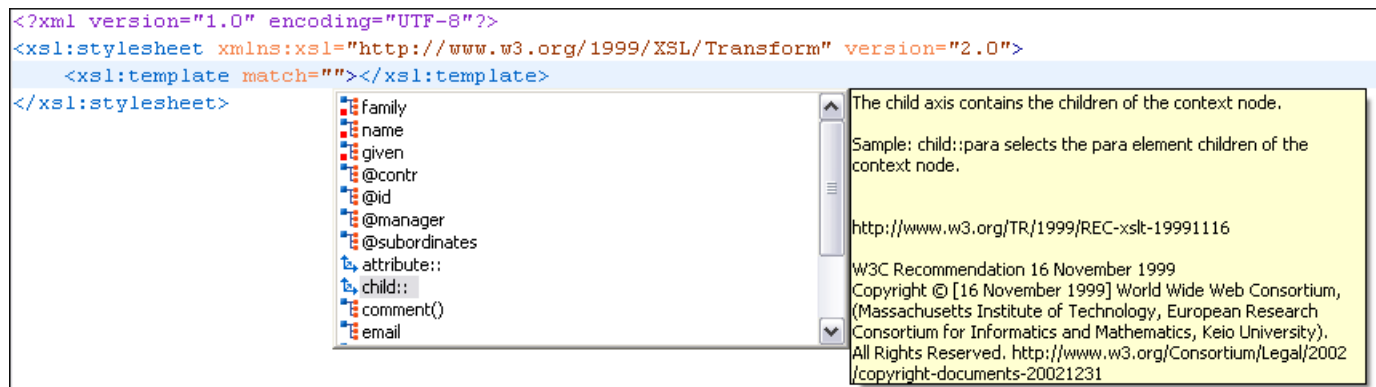


Figure 96: Content Completion in the `match` Attribute

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:

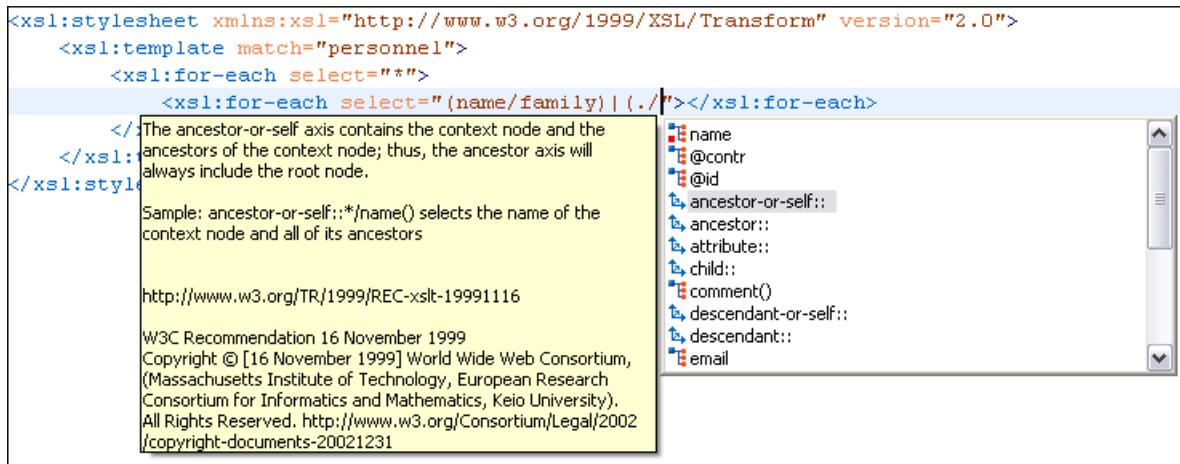


Figure 97: Content Completion in the select Attribute

Also XPath expressions typed in the `test` attribute of an `xsl:if` or `xsl:when` element benefit of the assistance of the content completion.

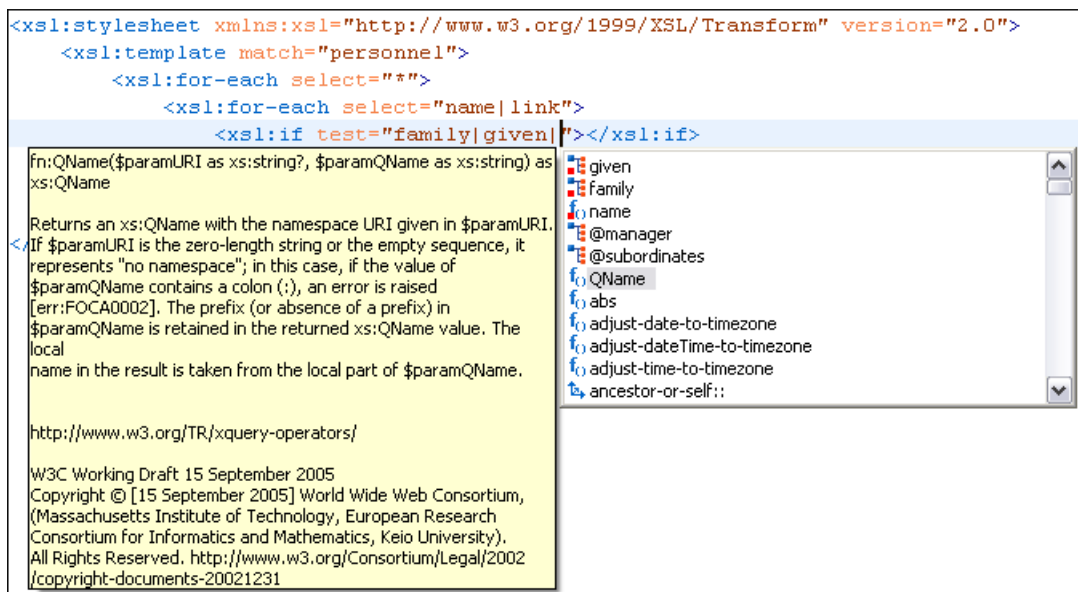


Figure 98: Content Completion in the test Attribute

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 99: 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.

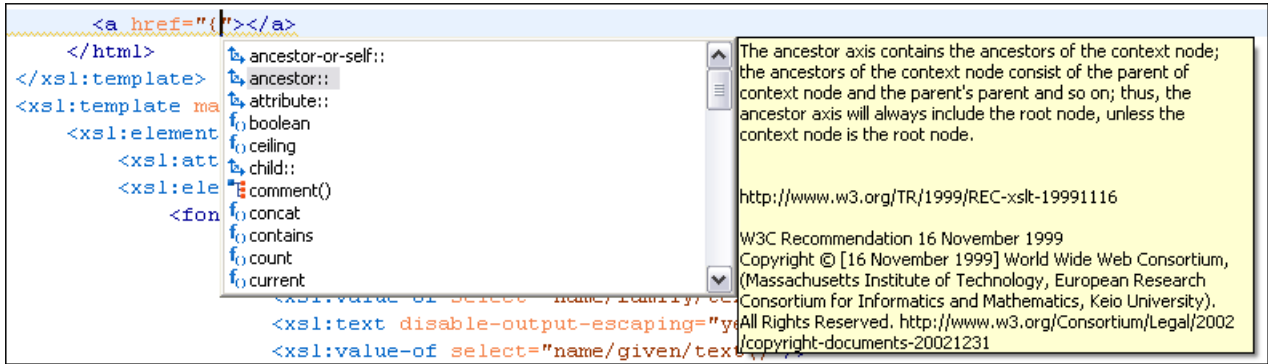


Figure 100: Content Completion in Attribute Value Templates

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 plugin 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 plugin 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?, $arg2 as xdt:anyAtomicType?, ...) as xs:string
match:
select: "concat(abs($v1), abs($v2))" </xsl:value-of>
</xsl:template>
!stylesheet
```

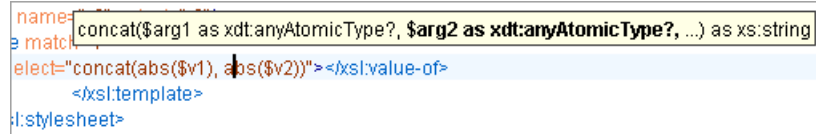
Figure 101: XPath Tooltip Helper - Identify the `concat` Function's First Argument

Moving the caret on the first variable `$v1`, the editor identifies the `abs` as context function and shows its signature:

```
name="v2" select="concat(abs($arg as numeric?) as numeric?"
match="/">
select="concat(abs($v1), abs($v2))" </xsl:value-of>
</xsl:template>
!stylesheet
```

Figure 102: 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?, $arg2 as xdt:anyAtomicType?, ...) as xs:string
match:
select="concat(abs($v1), abs($v2))" => </xsl:value-of>
</xsl:template>
:stylesheet>

```

Figure 103: 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.

Code Templates

When the content completion is invoked by pressing **(CTRL (Meta on Mac OS)+Space)**, it also presents a list of code templates specific to the type of the active editor. Such a code template provides a shortcut for inserting a small document fragment at the current caret position. Oxygen XML Editor plugin comes with a large set of ready-to use templates for XSL and XML Schema documents.

The XSL code template called Template-Match-Mode

Typing `t` in an XSL document and selecting `tmm` in the content assistant pop-up window inserts the following template at the caret position in the document:

```

<xsl:template match="" mode="">
</xsl:template>

```

The user *can easily define other templates*. Also, the code templates *can be shared with other users*.

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 referring to the dragged tree node. This value is based on the current editing context of the drop spot.

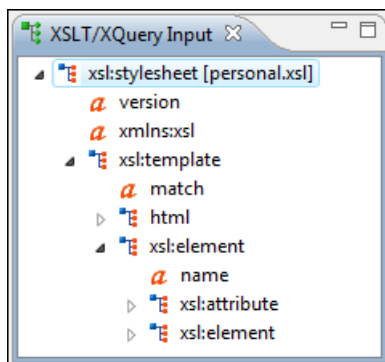


Figure 104: XSLT Input View

For example, for the following XML document:

```
<personnel>
  <person id="Big.Boss">
    <name>
      <family>Boss</family>
      <given>Big</given>
    </name>
    <email>chief@oxygenxml.com</email>
    <link subordinates="one.worker"/>
  </person>
  <person id="one.worker">
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>
    <email>one@oxygenxml.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 105: XSLT Input Drag and Drop Popup Menu

Select for example **Insert xsl:value-of** and the result document is:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
  <xsl:template match="personnel">
    <xsl:for-each select="*">
      <xsl:value-of select="name/given"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Figure 106: 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) 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 > Other > oXygen > Outline**.

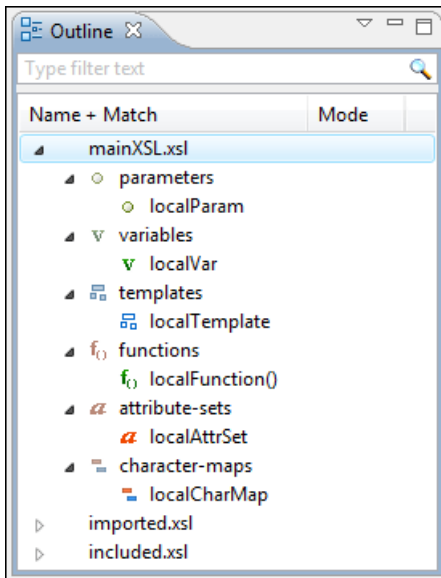




Figure 107: The XSLT Outline View

The following actions are available in the **View menu** on the Outline view action bar:

- **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 (Meta on Mac OS)+Shift+R)** - 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** - 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 the 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 plugin 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 plugin 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 plugin sorts the components depending on their order in the document. Oxygen XML Editor plugin also takes into account the name of the file that the components are part of.

XSLT Stylesheet Documentation Support

Oxygen XML Editor plugin 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 referred to 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 plugin offers its own XML schema that defines such documentation elements. The schema is named `stylesheet_documentation.xsd` and can be found in `[oxygen-install-folder]/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 (Meta on Mac OS) + Space**, 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 plugin 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 plugin 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, go to **XML Tools > Generate Documentation > XSLT Stylesheet Documentation...** action. You can also select **Generate Stylesheet Documentation** in the contextual menu of the **Navigator**.

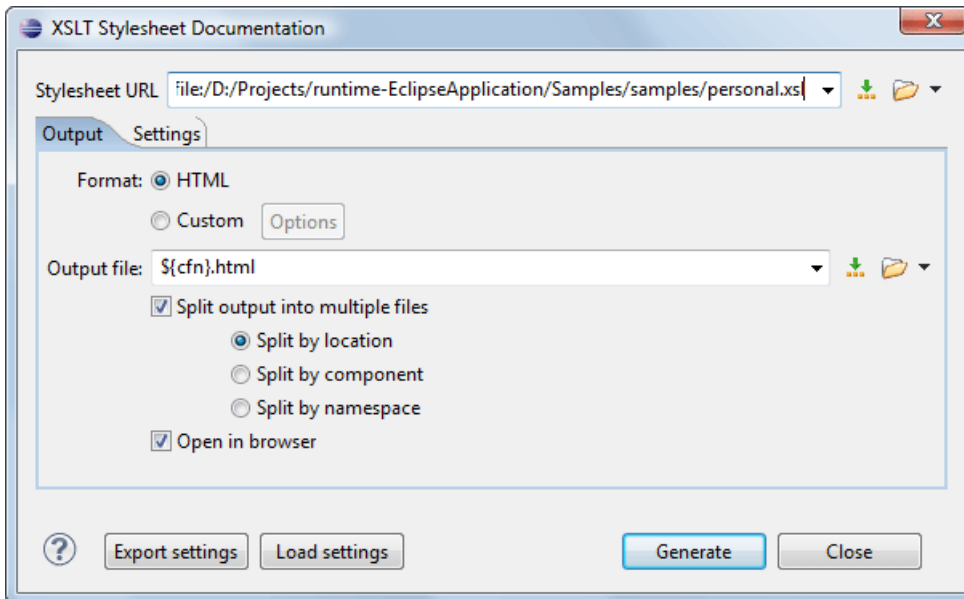


Figure 108: The Output Panel of the XSLT Stylesheet Documentation Dialog

The **XSL URL** field of the dialog panel 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.

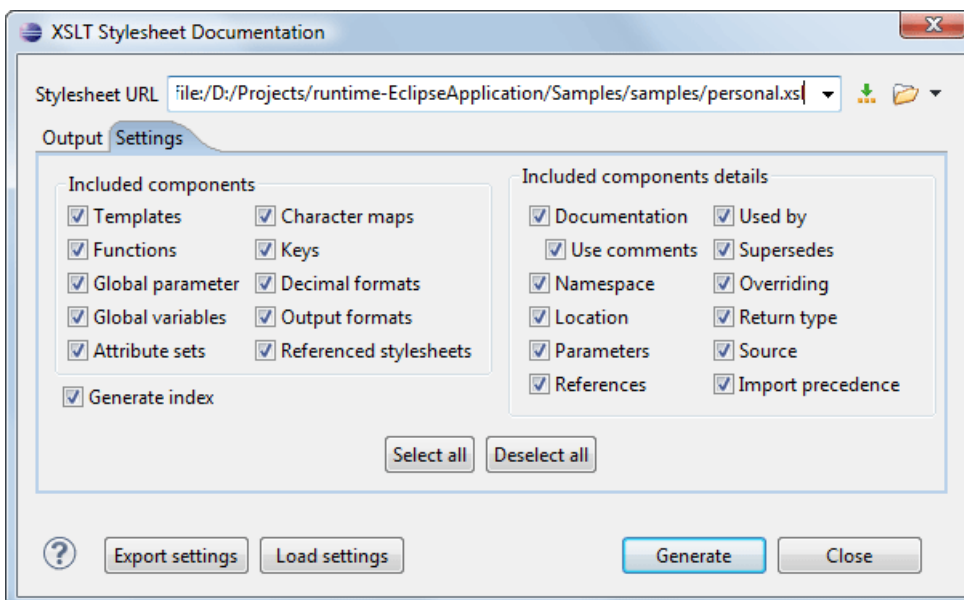


Figure 109: The Settings Panel of the XSLT Stylesheet Documentation Dialog

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 plugin 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`, `sl`, `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 referred from within an element.
- **Used by** - Shows the list of all the XSLT elements that refer 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:

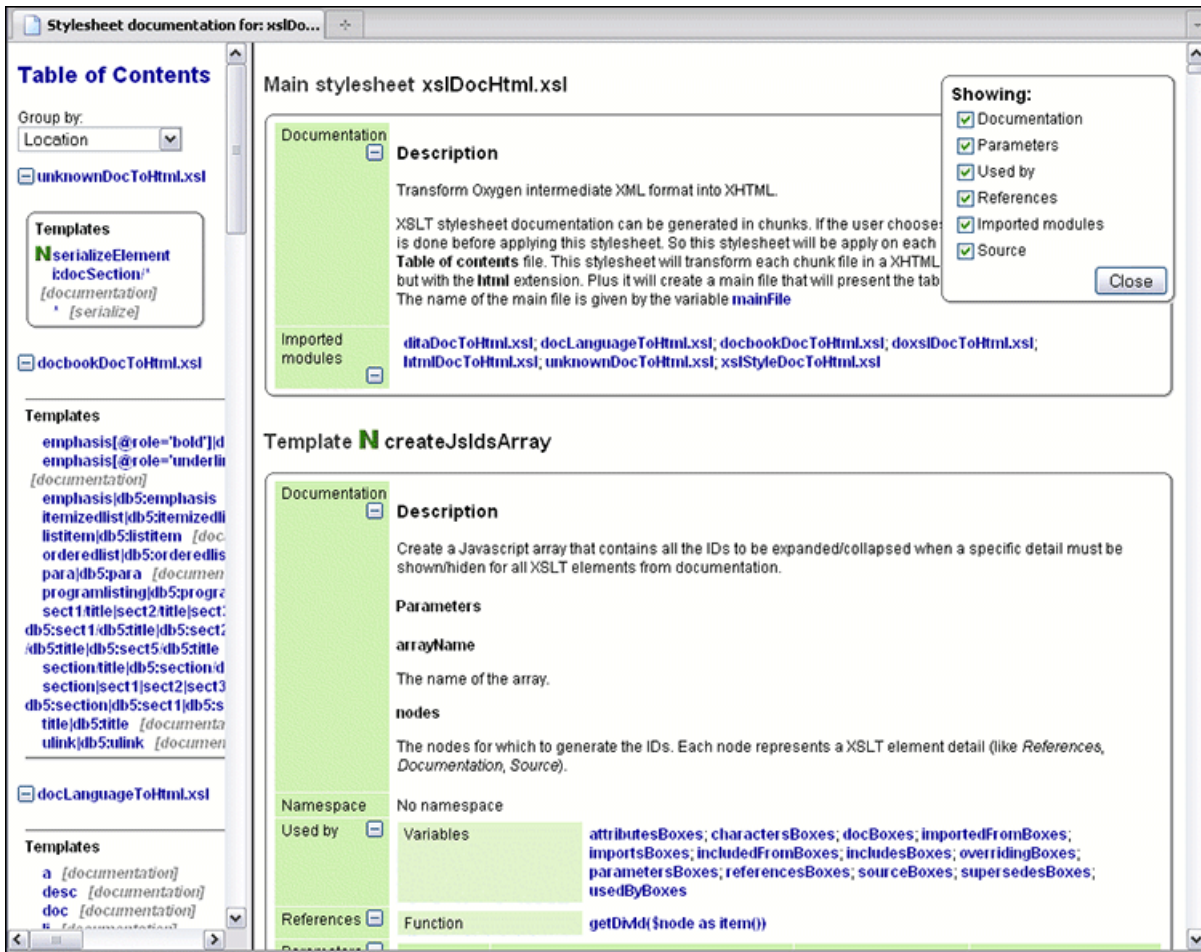


Figure 110: XSLT Stylesheet Documentation Example

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
 - and the stylesheet location.

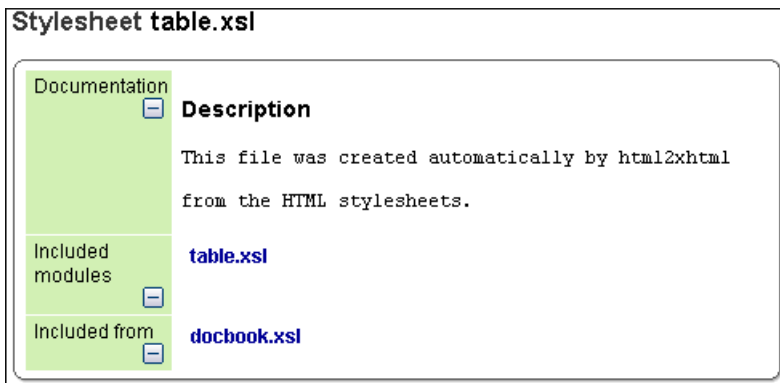


Figure 111: Information About an XSLT Stylesheet

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.

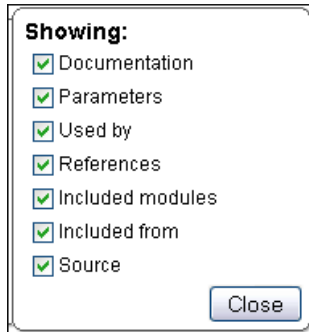


Figure 112: The Showing View

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	<p>Description</p> <p>Get the substring before the last occurrence of the given substring</p> <p>Parameters</p> <p>string The string in which to search</p> <p>searched The string to search</p> <p>Return The substring starting from the start of the string to the index of the last occurrence of searched</p>							
Namespace	http://www.oxygenxml.com/doc/xsl/functions							
Type	xs:string							
Used by	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Template</td> <td>Nindex</td> </tr> <tr> <td>Function</td> <td>func:substring-before-last(\$string as item(), \$searched as item())</td> </tr> <tr> <td>Variable</td> <td>indexFile</td> </tr> </table>	Template	Nindex	Function	func:substring-before-last(\$string as item(), \$searched as item())	Variable	indexFile	
Template	Nindex							
Function	func:substring-before-last(\$string as item(), \$searched as item())							
Variable	indexFile							
References	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Function</td> <td>substring-before-last(\$string as item(), \$searched as item())</td> </tr> </table>		Function	substring-before-last(\$string as item(), \$searched as item())				
Function	substring-before-last(\$string as item(), \$searched as item())							
Parameters	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>QName</th> <th>Namespace</th> </tr> </thead> <tbody> <tr> <td>searched</td> <td>No namespace</td> </tr> <tr> <td>string</td> <td>No namespace</td> </tr> </tbody> </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:concat(\$before,\$rec) </xsl:when> <xsl:otherwise> <xsl:string(\$string) </xsl:otherwise> </xsl:choose> </xsl:variable> <xsl:string(\$toReturn) </xsl:function></pre>							

Figure 113: 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. Specify your own stylesheet to transform the intermediary XML generated in the documentation process. You must write your stylesheet based on the schema `xslDocSchema.xsd` from `[Oxygen-install-folder]/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-install-folder]/frameworks/stylesheet_documentation/xsl`.

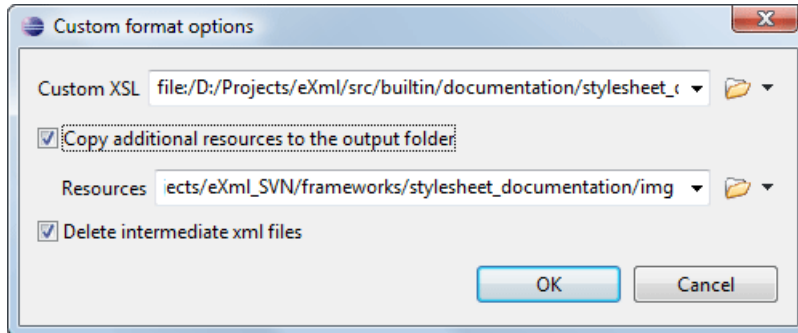


Figure 114: The Custom Format Options Dialog

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 From the Command Line Interface

You can export the settings of the **XSLT Stylesheet Documentation** dialog 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 Mac OS X / Unix / Linux) located in the Oxygen XML Editor plugin 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>
      </xsdDocumentationOptions>
    </entry>
  </map>
</serialized>
```



```

<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 actions are available for search operations related with XSLT references and declarations:

- **XSL** >  > **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 determined resources, a warning dialog is shown. This dialog allows you to define another search scope.
- **contextual menu of current editor** > **Search** > **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 dialog above.
- **XSL** >  **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 is shown.
- **contextual menu of current editor** > **Search** > **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when define a new scope.
- **XSL** > **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **XSL** > **Show Definition** - Moves the cursor to the location of the definition of the current item.



Note: You can also use the **Ctrl (Meta on Mac OS) + Click** 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 plugin 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 plugin 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 set on automatic search by default and can be configured in the **Options > Preferences > Editor > Mark Occurrences** page. A search can also be triggered with the **Search > Search Occurrences in File (Ctrl (Meta on Mac OS)+Shift+U)** contextual menu action. Matches are displayed in separate tabs of the **Results** view.

XSLT Refactoring Actions

The following actions allow changing the structure of an XSLT stylesheet without changing the results of running it in an XSLT transformation:


- **XSL** >  > **Create Template from Selection...** - 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 referring the newly created template.



Note: The selection must contain well-formed elements only.



Note: The newly created template is indented and its name is highlighted in the `<xsl:call-template>` element.

- **XSL** >  > **Create Stylesheet from Selection...** - Creates a separate stylesheet and replaces the selection with a `<xsl:include>` instruction referring the newly created stylesheet.



Note: The selection must contain a well-formed top-level element.

- **XSL** > **Extract Attributes as xsl:attributes...** - Extracts the attributes from the selected element and represents each of them with a `<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>
```

- **contextual menu of current editor** > **Refactoring** >  **Rename Component...** - Renames the selected component. Specify the new name for the component and the files affected by the modification as described for *XML Schema*.

To watch our video demonstration about XSLT refactoring, go to http://oxygenxml.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** > **Other** > **oXygen** > **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.

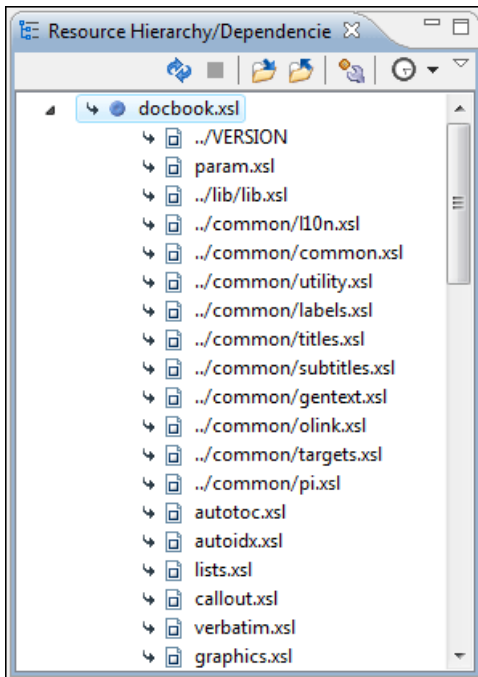


Figure 115: Resource Hierarchy/Dependencies View - Hierarchy for docbook.xsl

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.

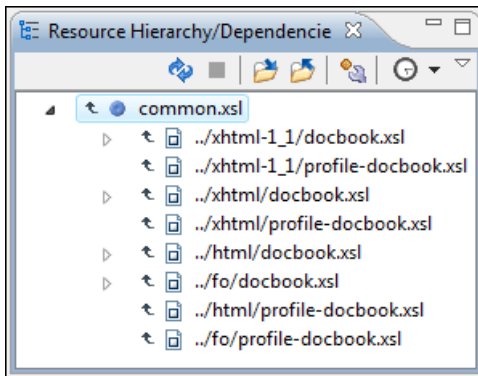











Figure 116: Resource Hierarchy/Dependencies View - Dependencies for common.xsl

The following actions are available in the **Resource Hierarchy/Dependencies** view:

-  - Refreshes the Hierarchy/Dependencies structure;
-  - Stop the hierarchy/dependencies computing;
-  - Allows you to choose a resource to compute the hierarchy structure;
-  - Allows you to choose a resource to compute the dependencies structure;
-  - 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;
-  - Repeats a previous dependencies computation.

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 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 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 is available for both actions. The **Preview** dialog 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 > Other > oXygen > 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, etc).

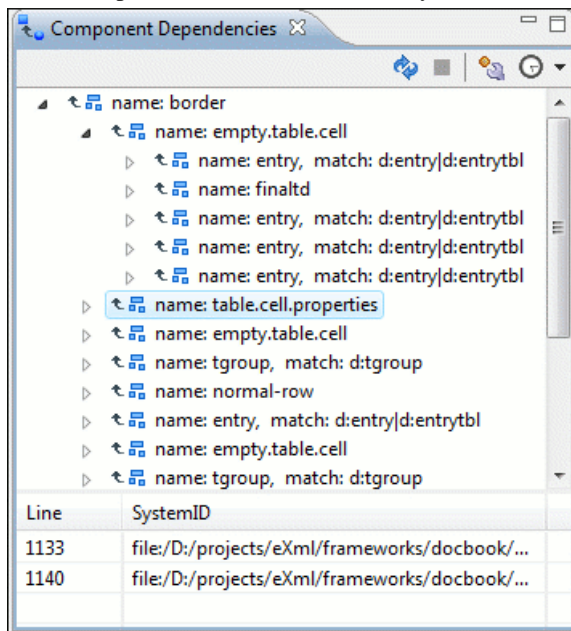






Figure 117: Component Dependencies View - Hierarchy for table.xsl

In the Component Dependencies view you have several actions in the toolbar:

-  - Refreshes the dependencies structure.
-  - Stops the dependencies computing.
-  - 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.
-  - 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 referred 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

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 **Ctrl + 1** keys (**Meta 1** on Mac OS X) on your keyboard.

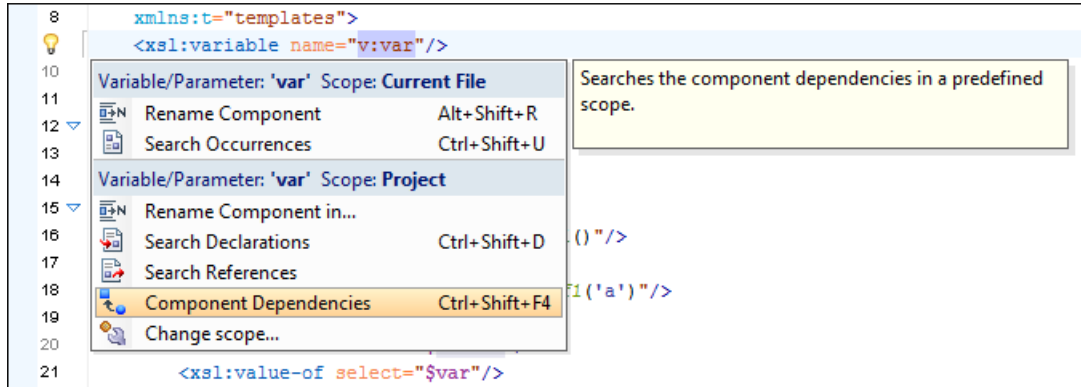


Figure 118: XSLT Quick Assist Support

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 in-place editing of the current component. When you use in-place editing, the component and all its references in the document are decorated with a thin border. All the changes you make to the component at caret position are updated in real time to all component's occurrences. To exit in-place editing, press either the **Esc** or **Enter** key on your keyboard;
- **Search Occurrences** - Searches all occurrences of the component within the current file.


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 plugin 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 plugin uses information from the validation and transformation scenarios associated with the XSL file. From the transformation scenario Oxygen XML Editor plugin uses extensions and properties of Saxon 9.5.0.1, improving the ANT scenario associated with the XSpec document.

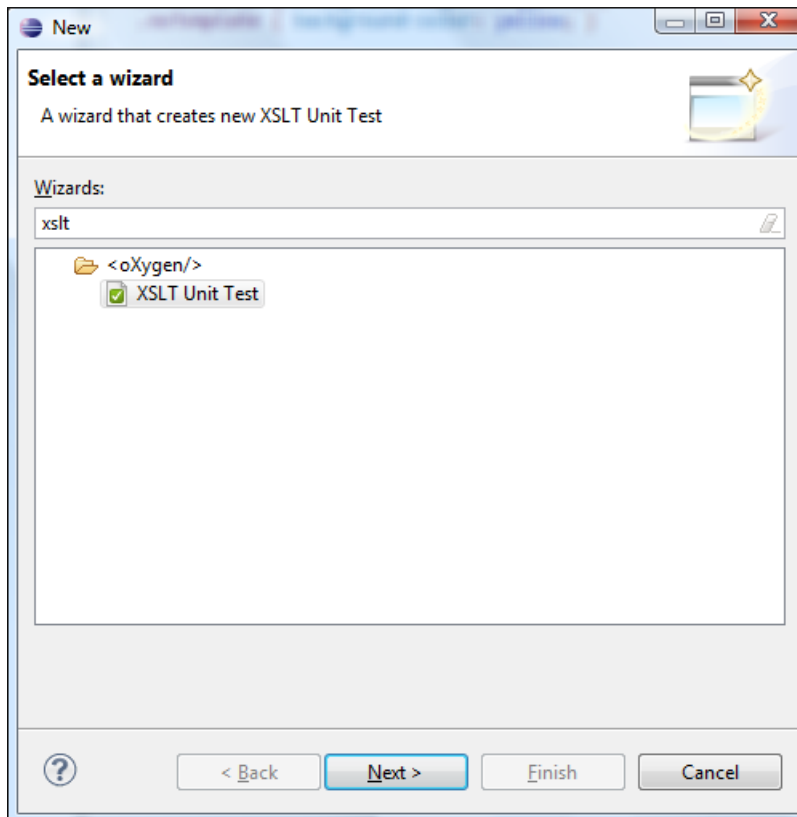


Figure 119: The New XSLT Unit Test wizard

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 refer 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 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 usual *Text* editing mode and the visual *Design* editing mode.

Oxygen XML Editor plugin 14.2 offers support both for XML schema 1.0 and 1.1.

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 plugin 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 Navigator 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 plugin 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;

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 *refactory 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.

Special Content Completion Features

The editor enhances *the content completion of the XML editor* inside the `xs:annotation/xs:appinfo` elements of an XML Schema with special support for the elements and attributes from a custom schema (by default ISO

Schematron). This content completion enhancement can be configured from the *XSD Content Completion* preferences page.

If the current XML Schema schema imports or includes other XML Schema schemas then the global types and elements defined in the imported / included schemas are available in the content completion window together with the ones defined in the current file.

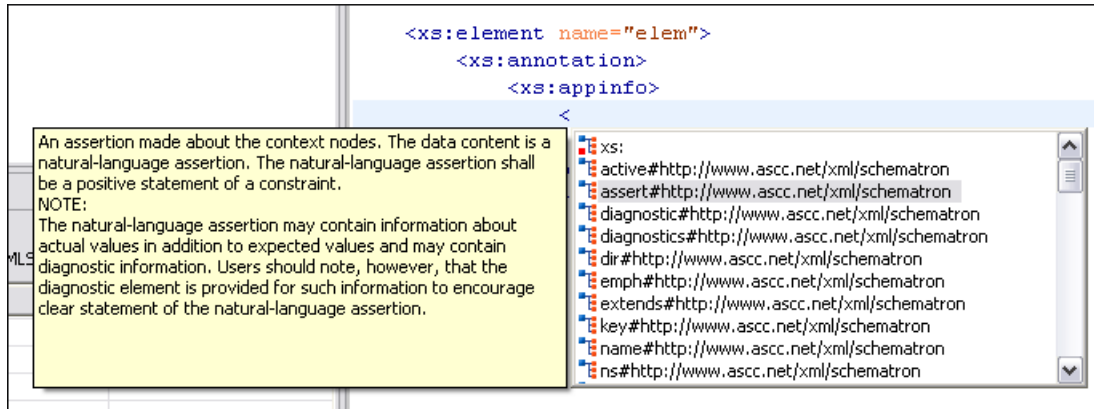


Figure 120: Schematron Support in XML Schema Content Completion

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 plugin calculates its hierarchy by processing the `xs:include` and `xs:import` statements.

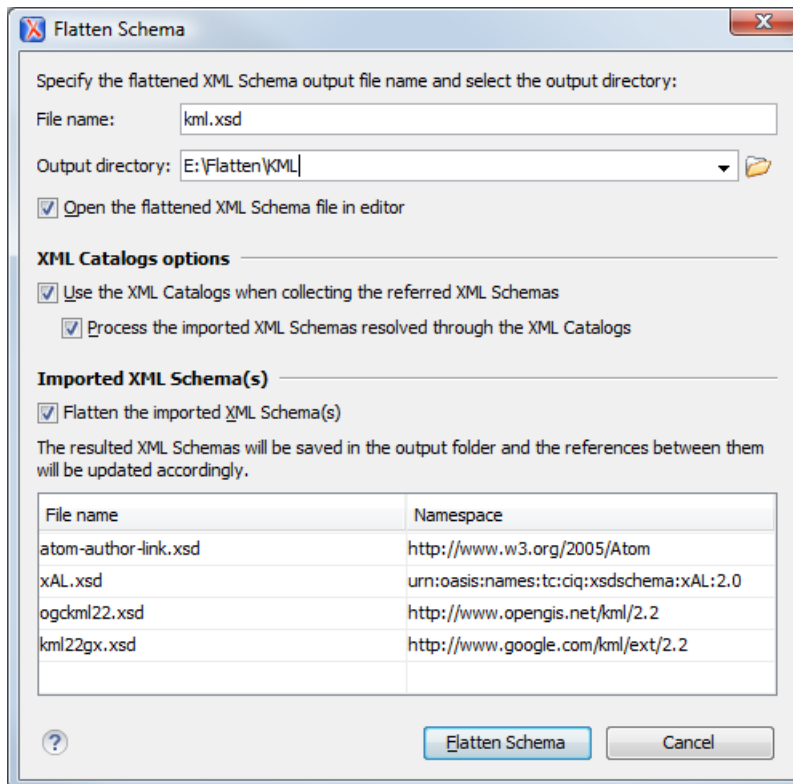



Figure 121: Flatten Schema Dialog

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 plugin 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 referred 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 plugin:

- `flattenSchema.bat` on Windows;
- `flattenSchema.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
```

Highlight Component Occurrences

When a component (for example types, elements, attributes) is found at current cursor position, Oxygen XML Editor plugin 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 set on automatic search by default and can be configured in the **Options > Preferences > Editor > Mark Occurrences** page. A search can also be triggered with the **Search > Search Occurrences in File > Ctrl (Meta on MAC OS) + Shift + U** contextual menu action. All matches are displayed in a separate tab of the **Results** view.

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 plugin 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.

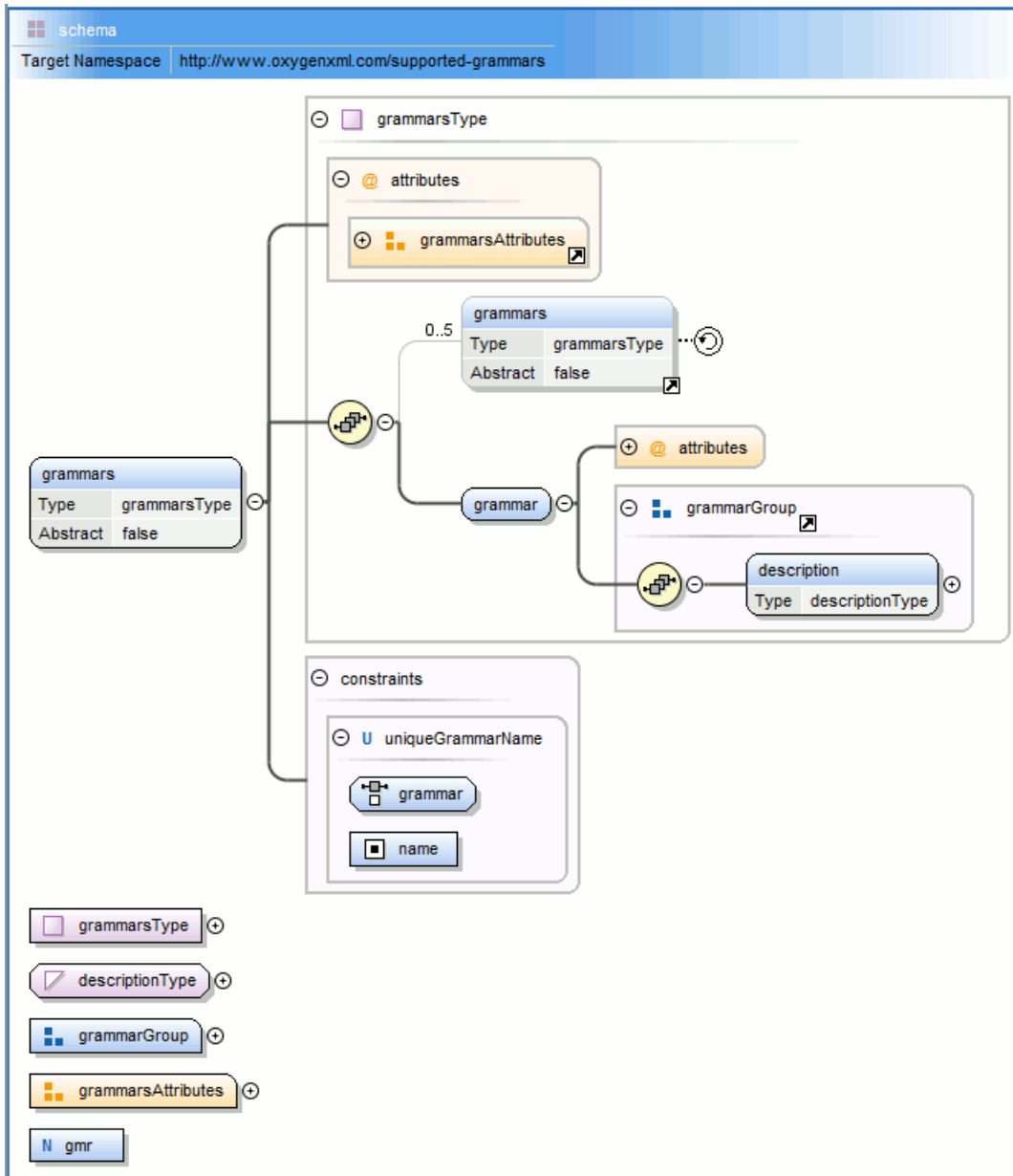


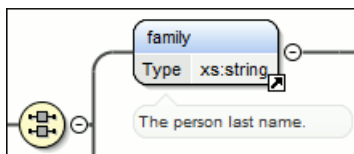
Figure 122: XML Schema Diagram

To watch our video demonstration about the basic aspects of designing an XML Schema using the new Schema Editor, go to http://oxygenxml.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

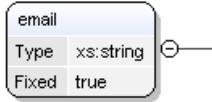
schema	
Target Namespace	http://www.oxygenxml.com/supported-grammars

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 referred 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 this property supports multiple values.	For global and reference elements
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
Nilable	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, 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 referred 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 referred 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 referred 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 <code>attributeFormDefault</code> attribute of the <code>xs:schema</code> 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



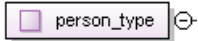
Defines an attribute group to be used in complex type definitions. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-attributeGroup>.

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 referred 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.

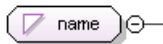
 **Tip:** A complex type which is a base type to another type will be rendered with yellow background.

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 <code>xsi:type</code> attribute.	true/false	For global and anonymous complex types.

Property Name	Description	Possible Values	Mentions
Block	Controls whether a substitution (either through a <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 "unlock" 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.

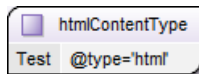
Name	Description	Possible Values	Scope
	(by list, union, derivation or all).	restriction union. In addition, [Empty] proposal is present for set empty string as value.	
ID	The component id.	Any id.	For all simple types
Component	The name of the edited component.	Not editable property.	Only for global 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.



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

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

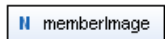
Property Name	Description	Possible Values
Schema Location	Redefine schema location.	Any URI
ID	Redefine ID	Any ID
Component	The component name.	Not editable property.

xs:override

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 123: 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 124: 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 125: 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 compositors.
Component	The edited component name.	Not editable property.	For all compositors.
System ID	The component system id.	Not editable property.	For all compositors.

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.

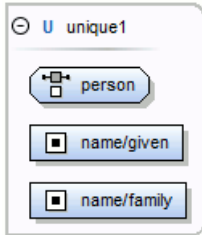
xs:anyAttribute



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

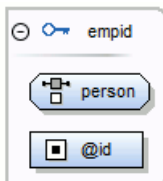
Property Name	Description	Possible Value
Namespace	The list of allowed namespaces. The namespace attribute expects a list of namespace URIs. In this list, two values have a specific meaning: '##targetNamespace' stands for the target namespace, and '##local' stands for local attributes (without namespaces).	##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

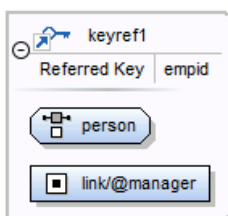
Property Name	Description	Possible Values
Name	The unique name. Always required.	Any NCName.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
Namespace	The component namespace.	Not editable property.
System ID	The component system id.	Not editable property.

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

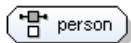
Specifies that an attribute or element value corresponds to that of the specified key or unique element. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-keyref>.

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.
Referred Key	The name of referred 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

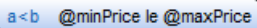
Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the field(s) composing the key, key reference, or unique constraint.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not 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.



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.



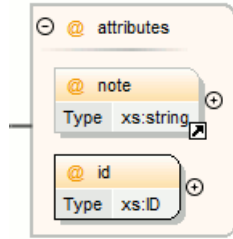
Note: This component is available for XML Schema 1.1 only. To change the version of the XML Schema, go to **Options > Preferences > XML > XML Parser > XML Schema**.

Constructs Used to Group Schema Components

This section explains the components that can be used for grouping other schema components:

- *Attributes*
- *Constraints*
- *Substitutions*

Attributes

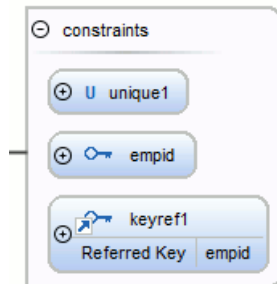


Groups all attributes and attribute groups belonging to a complex type.

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

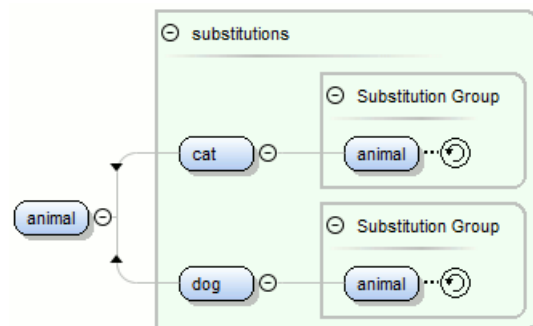


Groups all constraints (*xs:key*, *xs:keyRef* or *xs:unique*) belonging to an element.

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/refer 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 (Meta on Mac OS) + Click**.
 - 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 (Meta on Mac OS) + Home** key combination to go to the diagram root and **Ctrl (Meta on Mac OS) + End** 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, refer 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.
 - 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 plugin.
-  **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 plugin [XML documents validation](#) capability.

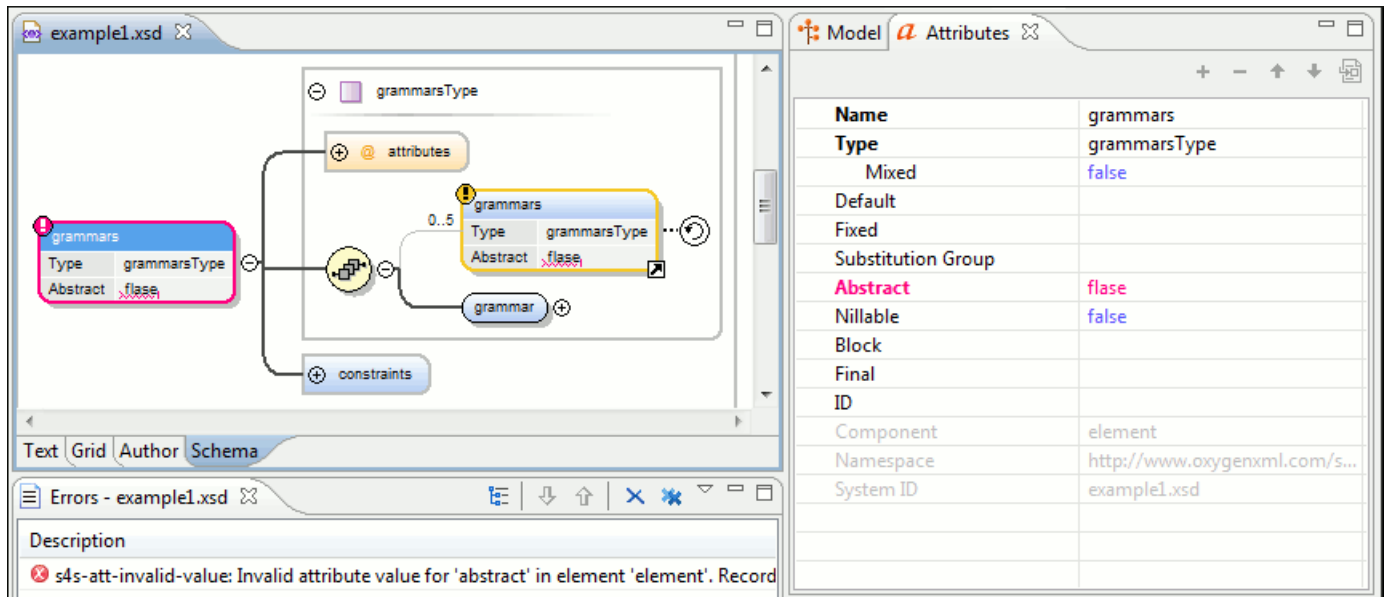


Figure 126: XML Schema Validation

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;

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 **Document > Validate Document (Alt+Shift+V) ((Cmd+Alt+V on Mac OS))** or the action available on the **Validate** toolbar, 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.

i **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;
- **refer** - Context dependent, the selected component is referred 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 referring 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/define 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 (Meta on Mac OS) + Shift + ENTER)** - 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 (Meta on Mac OS) + Shift + ENTER)** - 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... (Alt - Shift - Enter)** - allows you to edit the attributes of the selected component in a dialog 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 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. See the **Edit Import** dialog for more details;



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. You can perform the following operations in the dialog:
 - **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.**  - 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  button. Also you can delete the selected item using the  button;
 - **Move items up/down** - to do this use the  and  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 referred component by double-clicking the annotations area. Otherwise you can edit the referred 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 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;

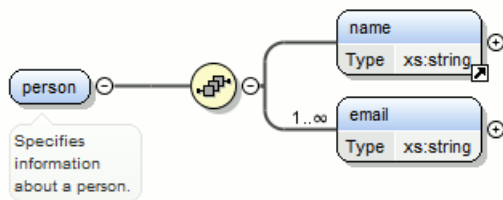
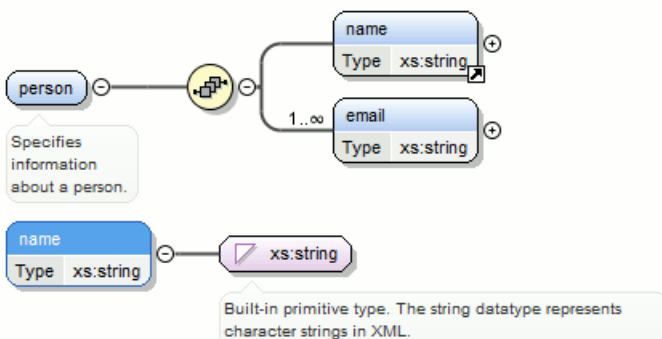


Figure 127: Extracting a Global Element

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;

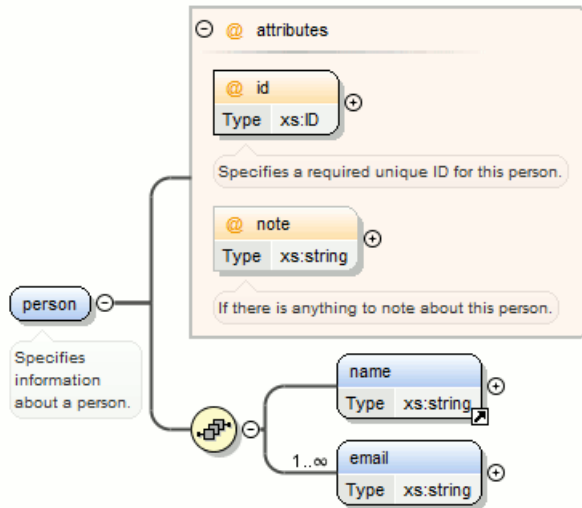
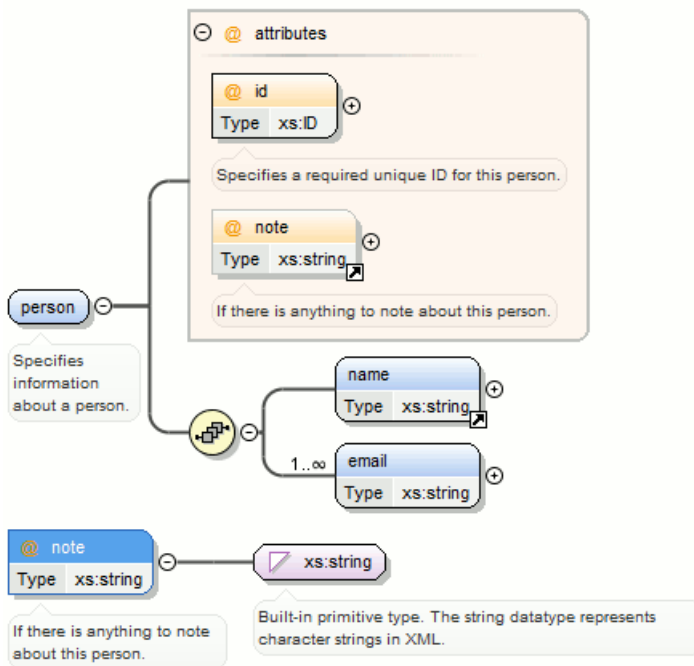
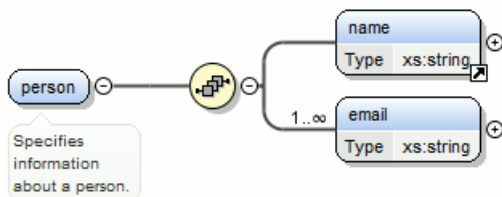


Figure 128: Extracting a Global Attribute

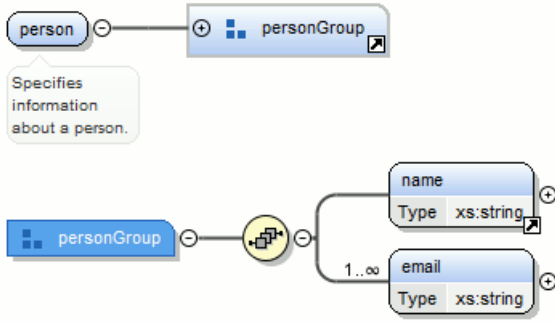
If you execute **Extract Global Attribute** on attribute `note` the result is:



- 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 is shown and you can choose a name for the group. If you type `personGroup`, the result



is:

Figure 129: 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;

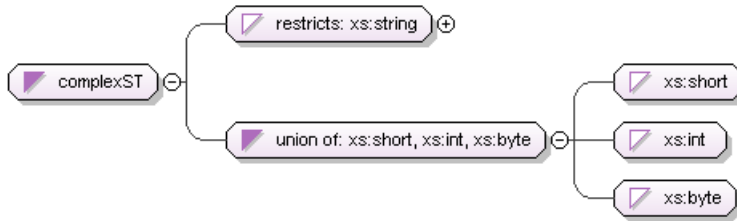


Figure 130: Extracting a Global Simple Type

If you use the action on the union component and choose `numericST` for the new global simple type name, the result is:

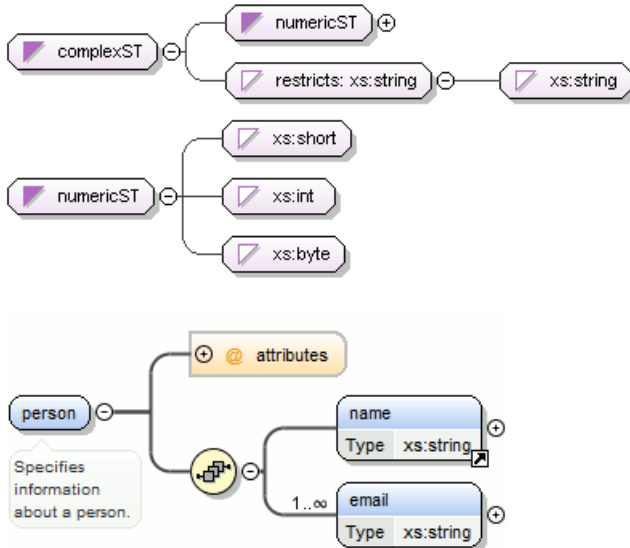
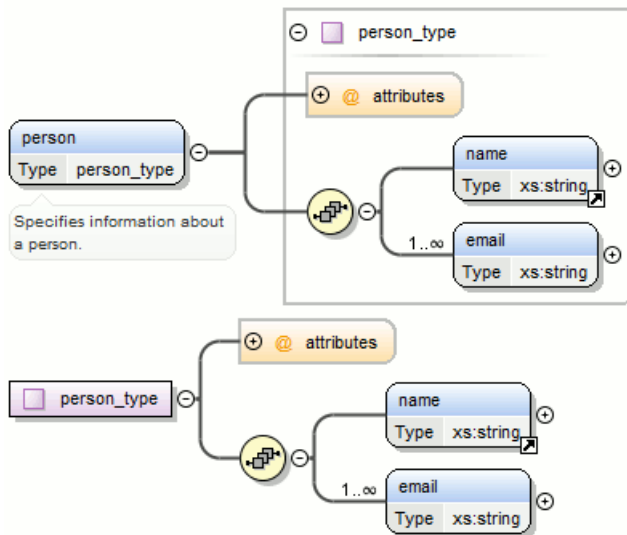











Figure 131: Extracting a Global Complex Type

If you execute the action on element `person` and choose `person_type` for the new complex type name, the result is:



- **Rename Component** - rename the selected component;
-  **Cut (Ctrl (Meta on Mac OS) - X)** - cut the selected component(s);
-  **Copy (Ctrl (Meta on Mac OS) - C)** - copy the selected component(s);
-  **Paste (Ctrl (Meta on Mac OS) - V)** -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 referred component is added in the current XML Schema. This option is available for *xs:redefine* 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 > Other > oXygen > Outline**.

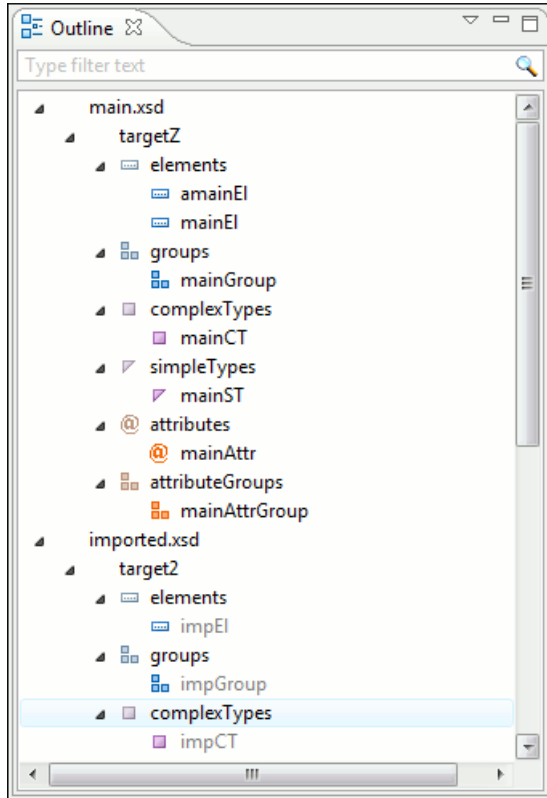








Figure 132: The Outline View for XML Schema

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 refer them, add an import directive with the *componentNS* namespace.
- **Show referable components** - Displays all components (collected starting from the main files) that can be referred 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** - 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** - Allows you to see the dependencies for the current selected component.

- **Resource Hierarchy** - Allows you to see the hierarchy for the current selected resource.
- **Resource Dependencies** - Allows you to see the dependencies for the current selected resource.
-  **Rename Component** - 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:

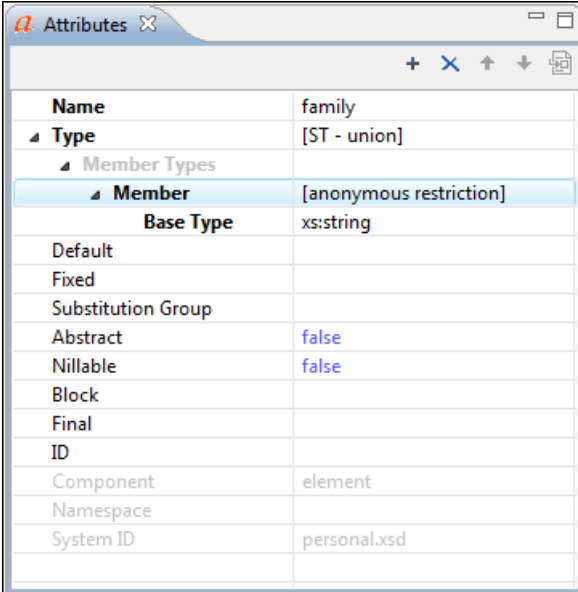
- * - any string
- ? - any character
- , - patterns separator

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 > Other > oXygen > Attributes**.



Name	Value
Name	family
Type	[ST - union]
Member Types	
Member	[anonymous restriction]
Base Type	xs:string
Default	
Fixed	
Substitution Group	
Abstract	false
Niltable	false
Block	
Final	
ID	
Component	element
Namespace	
System ID	personal.xsd

Figure 133: 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 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 > Other > oXygen > Facets**.

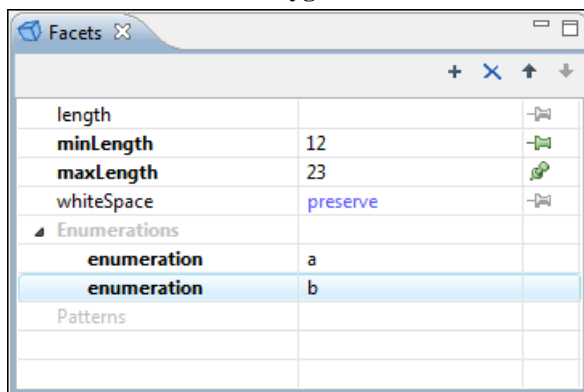



Figure 134: The Facets View

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 (eg: **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.

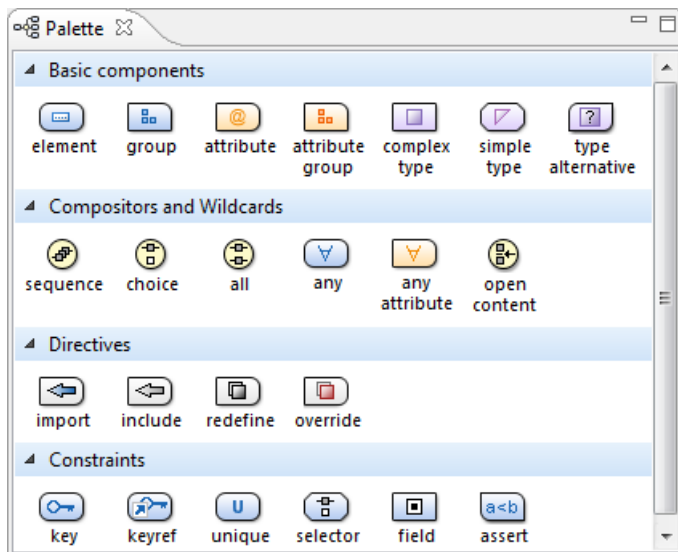
Editing Patterns

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 135: 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://oxygenxml.com/demo/Schema_Palette.html and http://oxygenxml.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.

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*.

Generate Sample XML Files

Oxygen XML Editor plugin 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 **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://oxygenxml.com/demo/Generate_Sample_XML_Files.html.

The Schema Tab

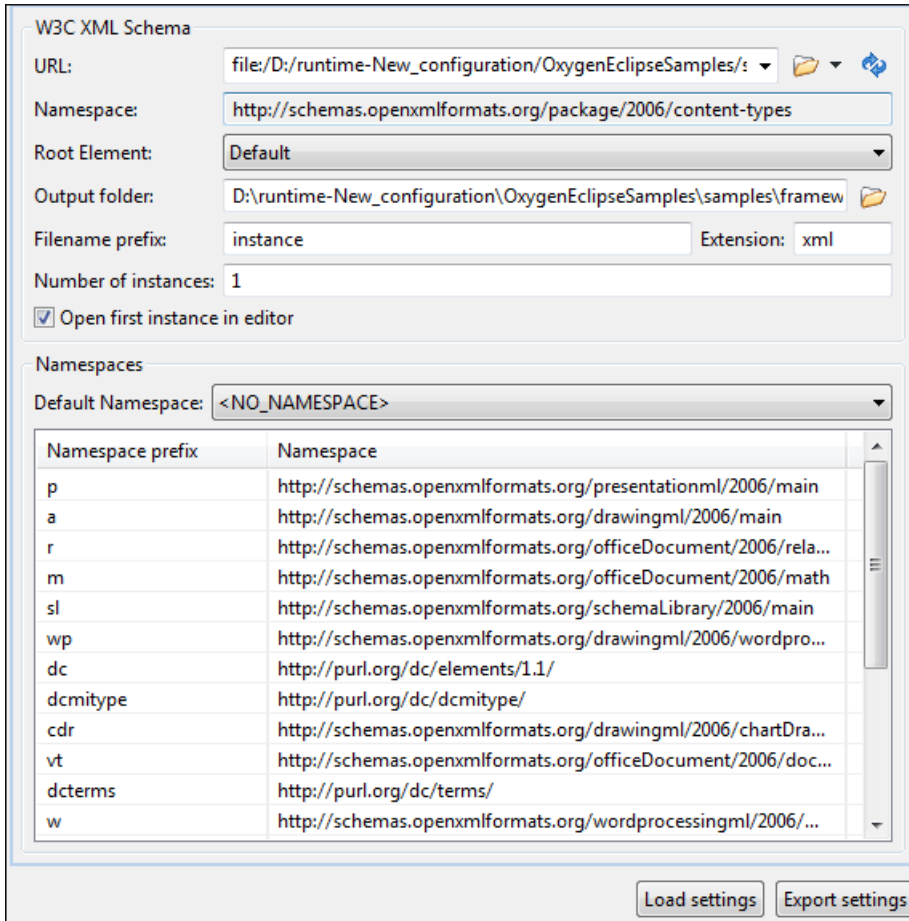


Figure 136: The Generate Sample XML Files Dialog

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.

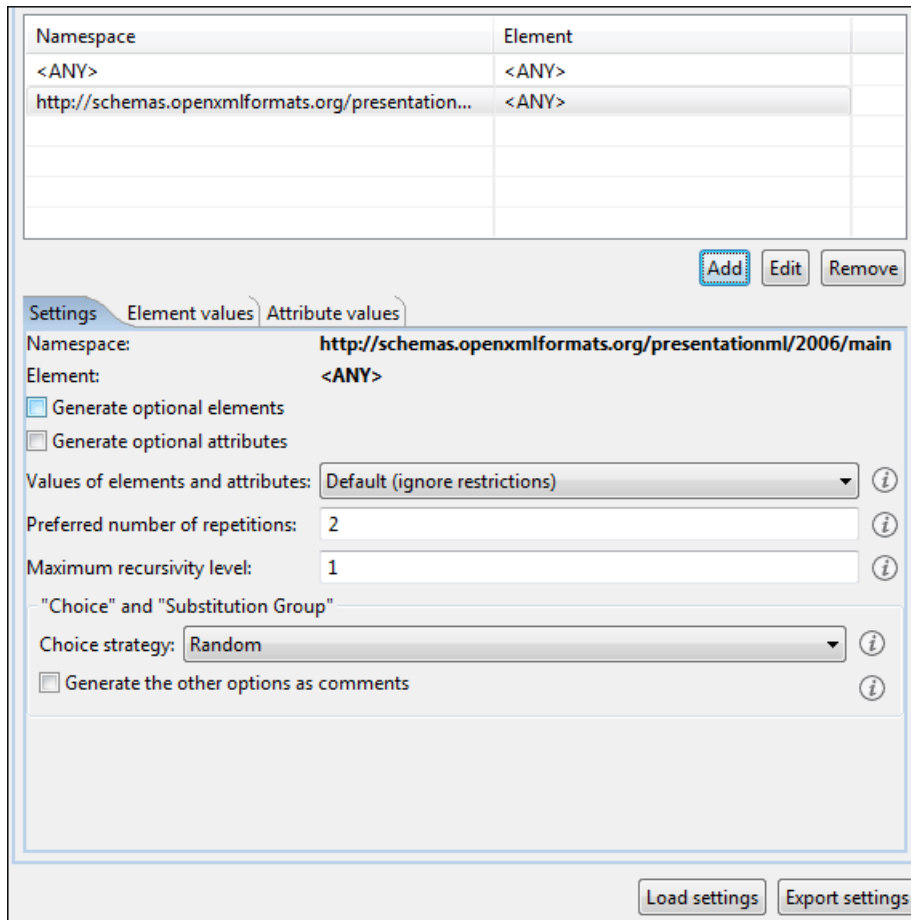


Figure 137: The Generate Sample XML Files Dialog

- **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

Strings and values	
<input checked="" type="checkbox"/>	Use incremental attribute/element names as default
Maximum length	30
Performance	
Discard optional elements after nested level	6

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.

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 menu **XML Schema Regular Expressions Builder**.

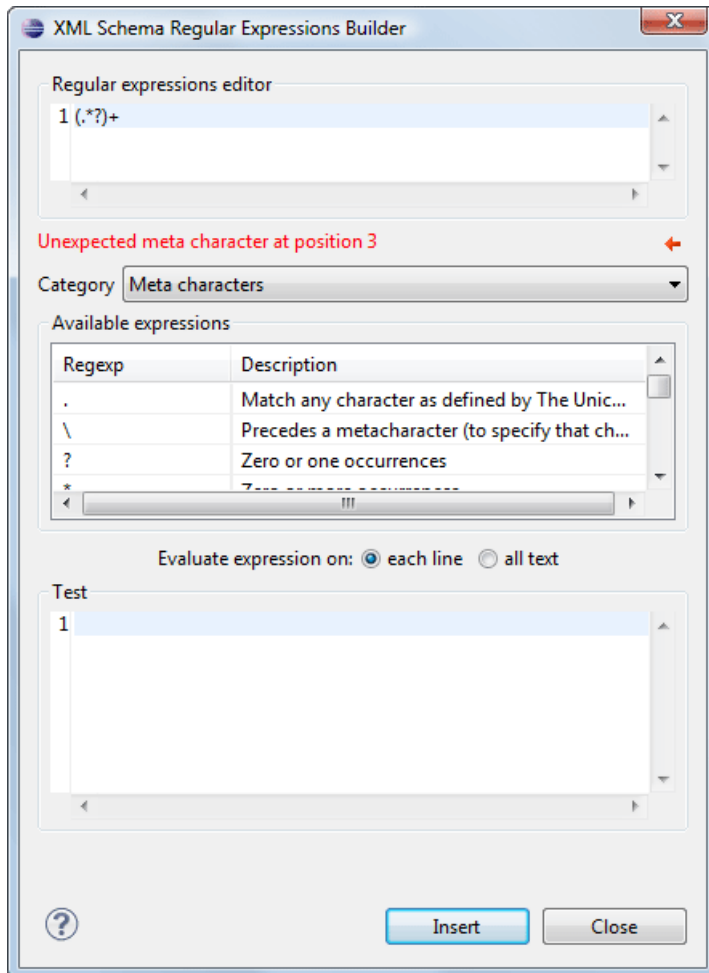


Figure 138: XML Schema Regular Expressions Builder Dialog


The dialog 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 (Meta on Mac OS) + Space**.
- **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: `p{Lu}` - Uppercase letters; `\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 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 that allows you to interrupt the operation.

Generating Documentation for an XML Schema

Oxygen XML Editor plugin 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, use the **Schema Documentation** dialog. Either go to **XML Tools > Generate Documentation > XML Schema Documentation...** or select **Generate XML Schema Documentation** from the contextual menu of the **Navigator** view. This dialog enables the user to configure a large set of parameters for the process of generating the documentation.

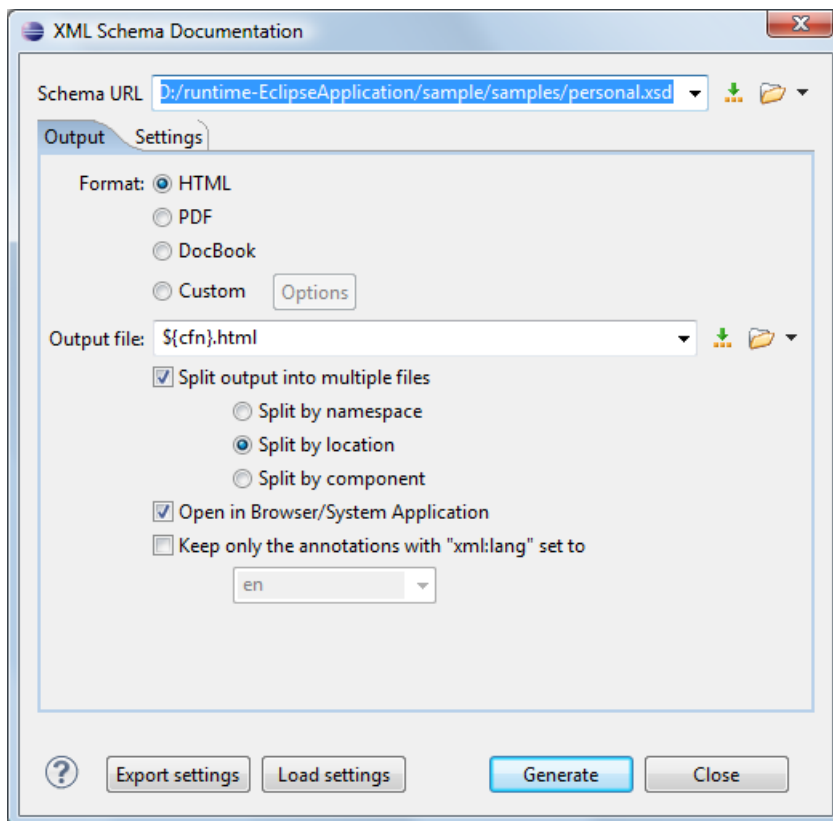


Figure 139: The Output panel of the XML Schema Documentation Dialog

The **Schema URL** field of the dialog panel 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: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes 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.

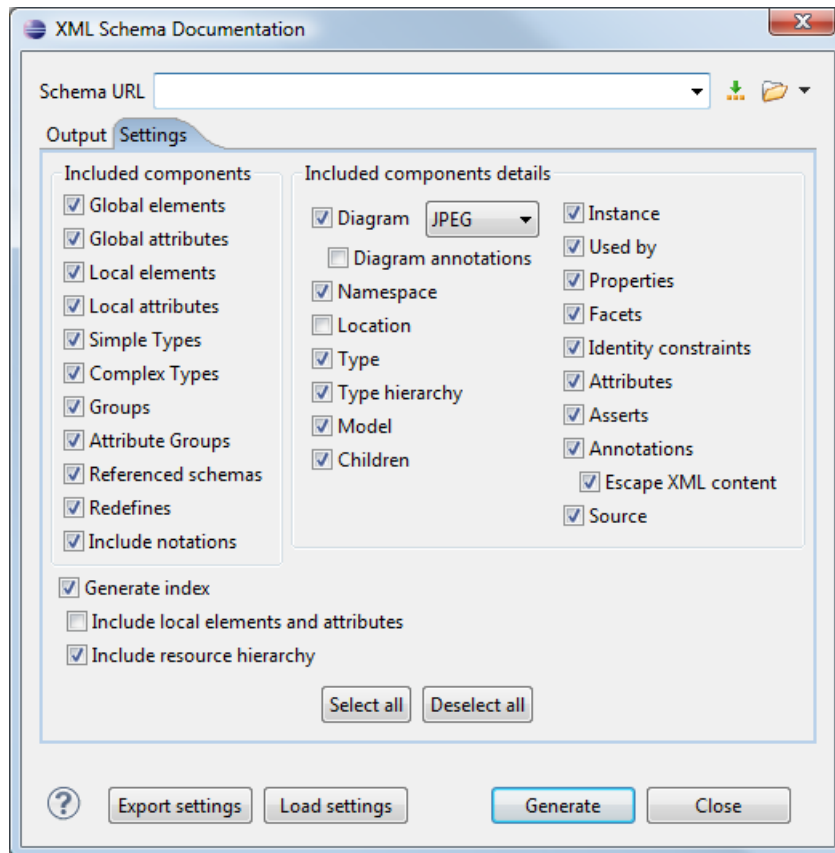


Figure 140: The Settings Panel of the XML Schema Documentation Dialog

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, 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 refer 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), refer 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 which refer to 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 plugin's XSD editor panel*.

Table of Contents

Group by: Location

personal.xsd

Elements

- email
- family
- given
- link
- name
- person
- personnel
- uri

Notations

- gif

XML Schema documentation generated by **Oxygen** XML Editor.

Element name

Namespace: No namespace

Annotations: Name, Annotation

Diagram: (Visual diagram showing a complex element 'name' containing 'family' and 'given' elements, with associated annotations and facets.)

Properties: Content: complex

Used by: Element: person

Model: ALL(family given)

Children: family, given

Instance:

```
<name>
<family>{1,1}</family>
<given>{1,1}</given>
</name>
```

Source:

```
<xs:element name="name">
<xs:annotation>
<xs:documentation>Name</xs:documentation>
<xs:documentation>Annotation</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:all>
<xs:element ref="family"/>
<xs:element ref="given"/>
</xs:all>
</xs:complexType>
</xs:element>
```

Showing:

- Annotations
- Attributes
- Diagrams
- Facets
- Identity Constraints
- Instances
- Model
- Properties
- Source
- Used by

Close

Figure 141: 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 142: 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:

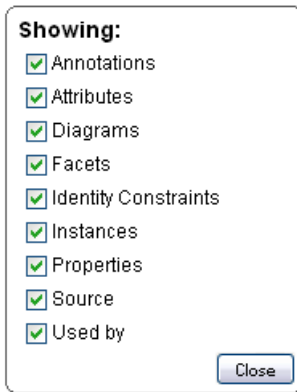


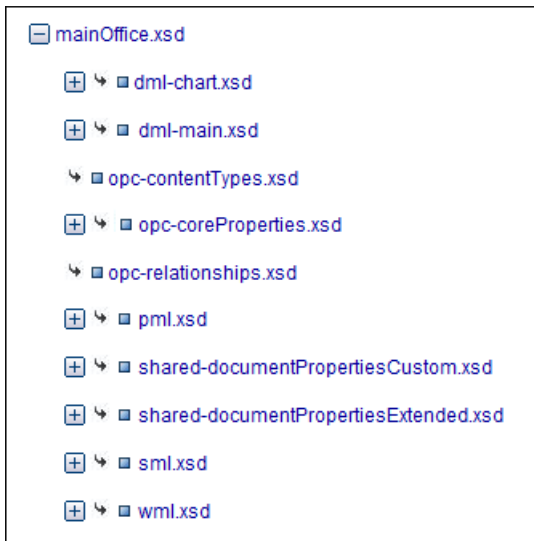
Figure 143: The Showing View

For each component included in the documentation, the section presents the component type followed 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.

Namespace	No namespace
Annotations	Specifies the person family and given name.
Diagram	
Properties	Content complex
Used by	Element person
Model	ALL(family given)
Children	family, given
Instance	<pre><name> <family>{1,1}</family> <given>{1,1}</given> </name></pre>
Source	<pre><xs:element name="name"> <xs:annotation> <xs:documentation>Specifies the person family and given name.</xs:documentation> </xs:annotation> <xs:complexType> <xs:all> <xs:element ref="family"/> <xs:element ref="given"/> </xs:all> </xs:complexType> </xs:element></pre>

Figure 144: 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. 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 plugin (*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 `{INSTALLATION_DIRECTORY}/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 `{INSTALLATION_DIRECTORY}/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, follow the next procedure:

- Customize the `[Oxygen-install-directory]/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.

- Create an intermediary file that holds the content of your XML Schema documentation;
- Go to **Tools > Generate Documentation > XML Schema Documentation**;

- Click **Custom** in the **XML Schema Documentation** dialog;
- Go to **Options**;
- In the **Custom format options** dialog enable **Copy additional resources to the output folder**, enter: `[Oxygen-install-directory]/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl` in the **Custom XSL** field and click **OK**;
- 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;
- Create the final PDF document;
 - Go to **Document > Transformation > Configure Transformation Scenario** and click **New**;
 - In the **New Scenario** dialog go to the **XSL URL** field and choose the `[Oxygen-install-dir]/frameworks/docbook/oxygen/xsdDocDocbookCustomizationFO.xsl` file;
 - Go to the **FO Processor** tab and enable **Perform FO Processing** and **XSLT result as input**;
 - Go to the **Output** tab and select the directory where you want to store the XML Schema documentation output and click **OK**;
 - Click **Apply Associated**.

Generating Documentation From the Command-Line Interface

You can export the settings of the **XML Schema Documentation** dialog 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 Mac OS X / Unix / Linux).

The scripts are located in the Oxygen XML Editor plugin 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>
      <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>
      </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>


```

Searching and Refactoring Actions

All the following actions can be applied on attribute, attributeGroup, element, group, key, unique, keyref, notation, simple, or complex types:

- **XSD** >  > **References (Alt+Shift+S R) ((Cmd+Alt+S R on Mac OS))** - 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 is shown and you have the possibility to define another search scope.

- **contextual menu of current editor > 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.
- **XSD >  > Declarations (Alt+Shift+S D) ((Cmd+Alt+S D on Mac OS))** - 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 will be shown and you have the possibility to define another search scope.

This action is not available in **Design** mode.

- **contextual menu of current editor > Search > Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when define a scope in the dialog above. Action is not available in **Design** mode.
- **XSD > Occurrences in File (Alt+Shift+S O) ((Cmd+Alt+S O on Mac OS))** - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor > Rename Component** - Allows in-place editing of the current component. When you use in-place editing, the component and all its references in the document are decorated with a thin border. All the changes you make to the component at caret position are updated in real time to all component's occurrences. To exit in-place editing, press either the **Esc** or **Enter** key on your keyboard.
- **contextual menu of current editor > Rename Component in...** - Renames the selected component. Specify the new component name and the file or files affected by the modification in the following dialog:

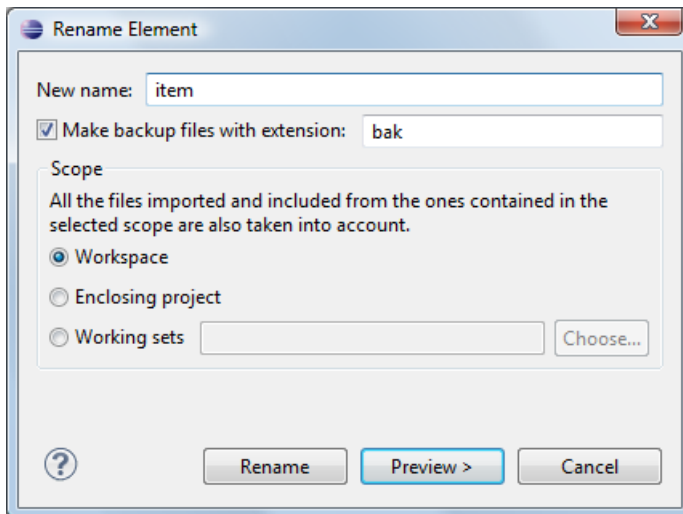


Figure 145: Rename Component Dialog

if you click the **Preview** button, you have the possibility to view the files affected by the **Rename Component** action. The changes are shown in the following dialog:

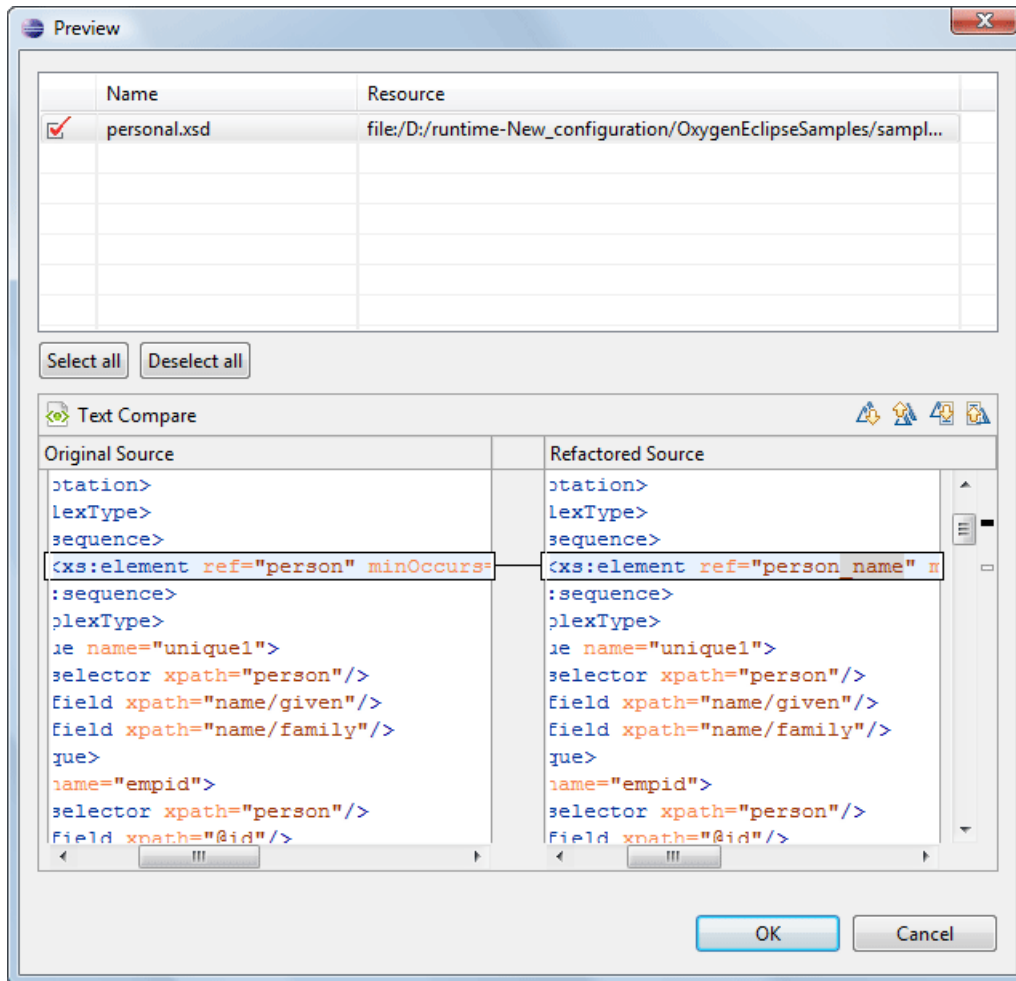


Figure 146: Preview Dialog

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 > Other > oXygen > 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.

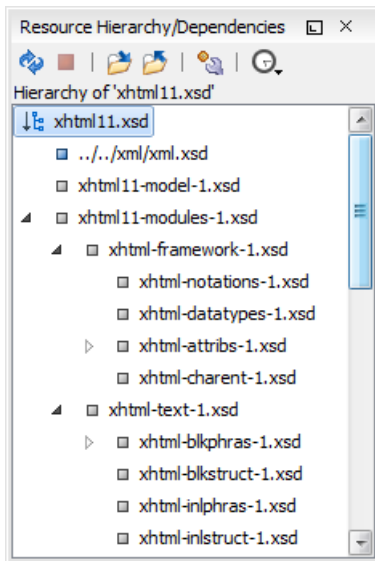


Figure 147: Resource Hierarchy/Dependencies View - Hierarchy for xhtml11.xsd

The build process for the dependencies view is started with the **Resource Dependencies** action available on the contextual menu.

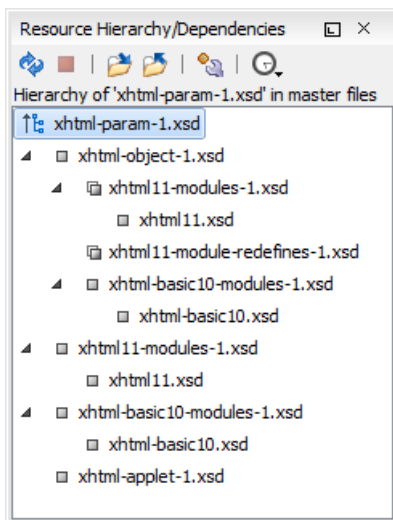









Figure 148: Resource Hierarchy/Dependencies View - Dependencies for xhtml-param-1.xsd

The following actions are available in the **Resource Hierarchy/Dependencies** view:

-  - Refreshes the Hierarchy/Dependencies structure;
-  - Stop the hierarchy/dependencies computing;
-  - Allows you to choose a resource to compute the hierarchy structure;
-  - Allows you to choose a resource to compute the dependencies structure;
-  - 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;
-  - Repeats a previous dependencies computation.

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 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 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 is available for both actions. The **Preview** dialog presents a list with the resources that are updated.

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 > Other > oXygen > 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).

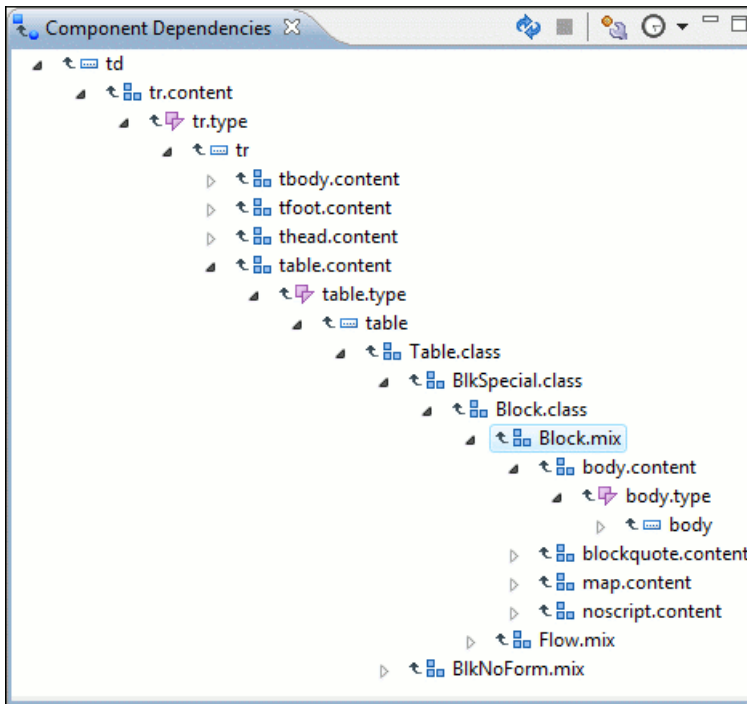


Figure 149: Component Dependencies View - Hierarchy for xhtml111.xsd

In the **Component Dependencies** view the following actions are available in the toolbar:

- - Refreshes the dependencies structure.
- - Stop the dependencies computing.
- - Allows you to configure a search scope to compute the dependencies structure.
- - 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 referred 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 Fix support improves the development work flow, offering fast access to the most commonly used actions when you edit 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 **Ctrl + 1** keys (**Meta 1** on Mac OS X) on your keyboard.

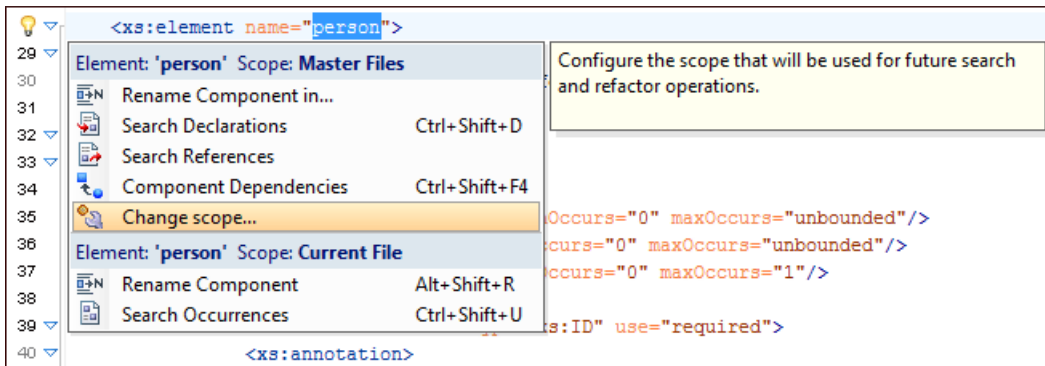


Figure 150: Quick Assist Support

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 in-place editing of the current component. When you use in-place editing, the component and all its references in the document are decorated with a thin border. All the changes you make to the component at caret position are updated in real time to all component's occurrences. To exit in-place editing, press either the **Esc** or **Enter** key on your keyboard;
- **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://oxygenxml.com/demo/Quick_Assist.html.

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.

XML Schema 1.1

Oxygen XML Editor plugin 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.

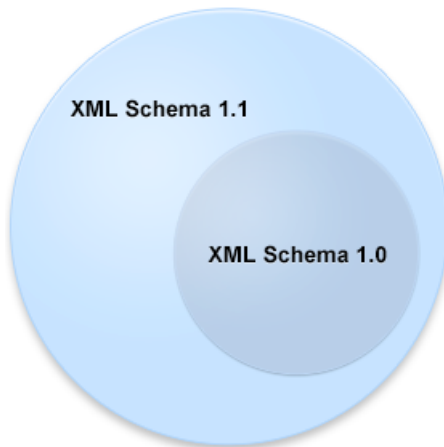



Figure 151: XML Schema 1.1

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 XML Schema 1.0. So even if you make a document invalid conforming with XML Schema 1.0, the validation process does not signal any issues.

You can change the default XML Schema version from **Options > Preferences > XML > XML Parser > XML Schema**.

To watch our video demonstration about the XML Schema 1.1 support, go to http://oxygenxml.com/demo/XML_Schema_11.html.

Setting the XML Schema Version

Oxygen XML Editor plugin 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 6: Using the `minVersion` and `maxVersion` Attributes to Set the XML Schema Version

Versioning Attributes	XML Schema Version
<code>minVersion = "1.0" maxVersion = "1.1"</code>	1.0
<code>minVersion = "1.1"</code>	1.1
<code>minVersion = "1.0" maxVersion = 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 plugin to its default options.

Oxygen XML Editor plugin 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 plugin takes into account the `minVersion` and `maxVersion` attributes defined in the XML Schema.

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 > Other > oXygen > Outline** menu action.

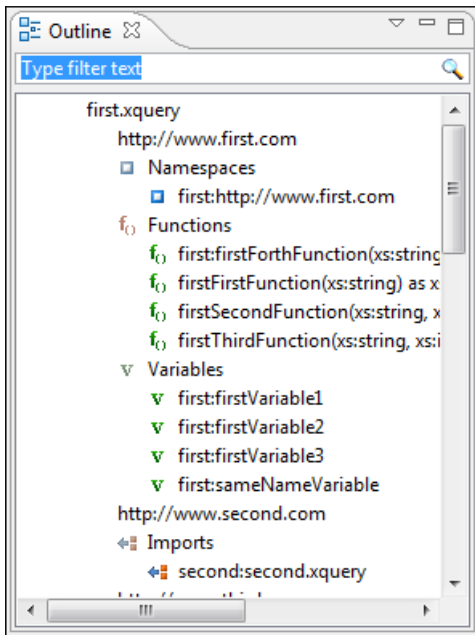


Figure 152: XQuery Outline View

The following actions are available in the **View** menu on the Outline view's action bar:

- **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.

```

let $minRating := min($review/reviews/review[@movie-id = $movie-id]/rating)
return
<movie id="{ $movie/@id }">
  { $movie/title }
  { $movie/year }
  <avgRating>
    {
      if ($avgRating) then $avgRating else "not rated"
    }
  </avgRating>
  <maxRating>
    <value>
      { }
    </value>
  </maxRating>
  <minRating>
    <value>
      { }
    </value>
  </minRating>
</movie>

```

Figure 153: Folding in XQuery Documents

There is available the action **Go to Matching Bracket** **Ctrl+Shift+G** 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.

Generating HTML Documentation for an XQuery Document

To generate HTML documentation for an XQuery document, use the dialog **XQuery Documentation**. It is opened with the action **XML Tools > Generate Documentation > XQuery Documentation....** . It can be also opened from the **Navigator's contextual menu > Generate XQuery Documentation**. The dialog enables the user to configure a set of parameters of the process of generating the HTML documentation. The parameters are:

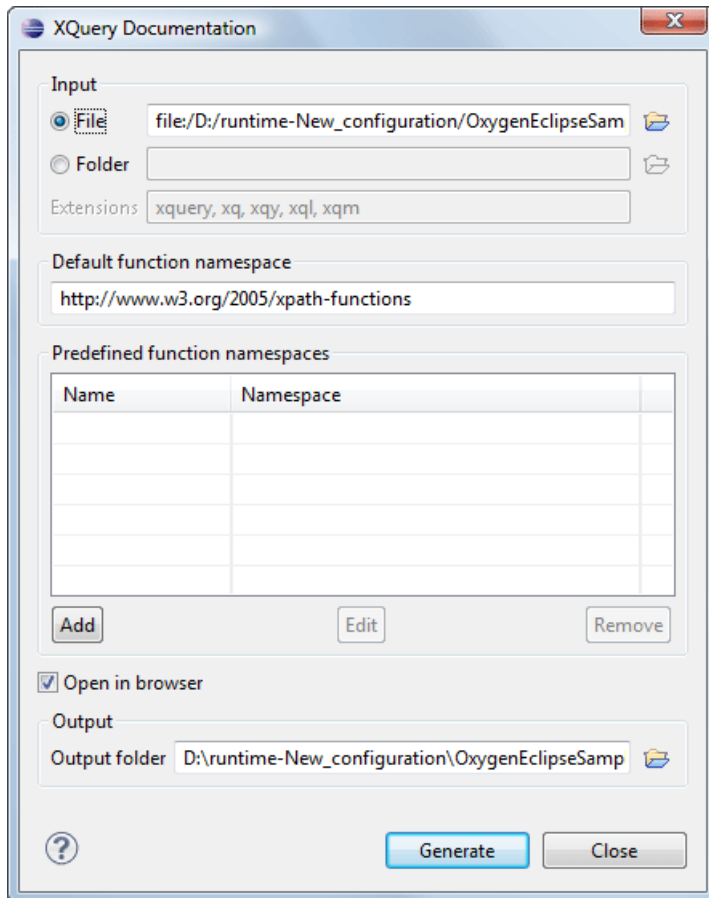


Figure 154: The XQuery Documentation Dialog

- **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: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes 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 plugin 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 plugin 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.

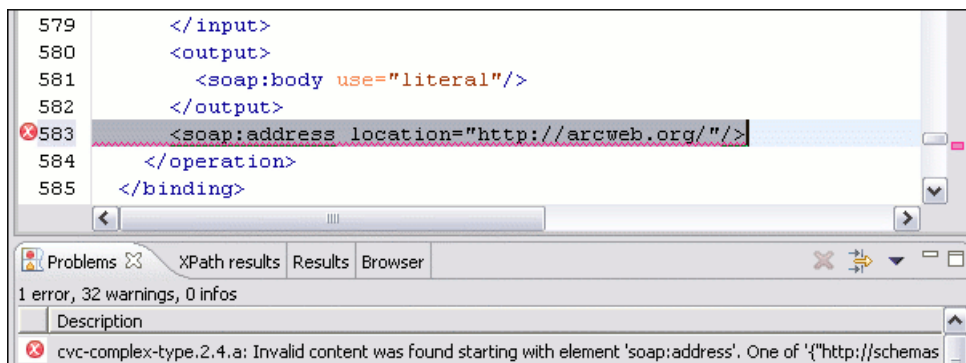


Figure 155: Validating a WSDL file

To watch our video demonstration about the WSDL editing support in Oxygen XML Editor plugin, 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 > Other > oXygen > Outline**.

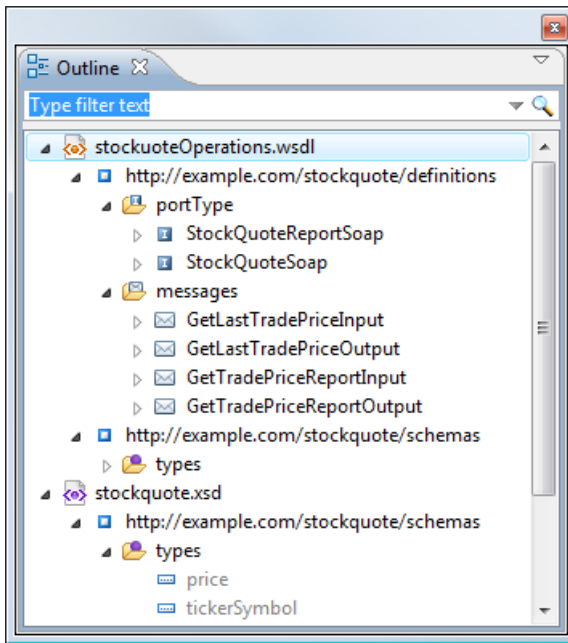











Figure 156: The WSDL Outline View



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 **View menu** on the Outline view action bar 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 refer 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.
- **T 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.

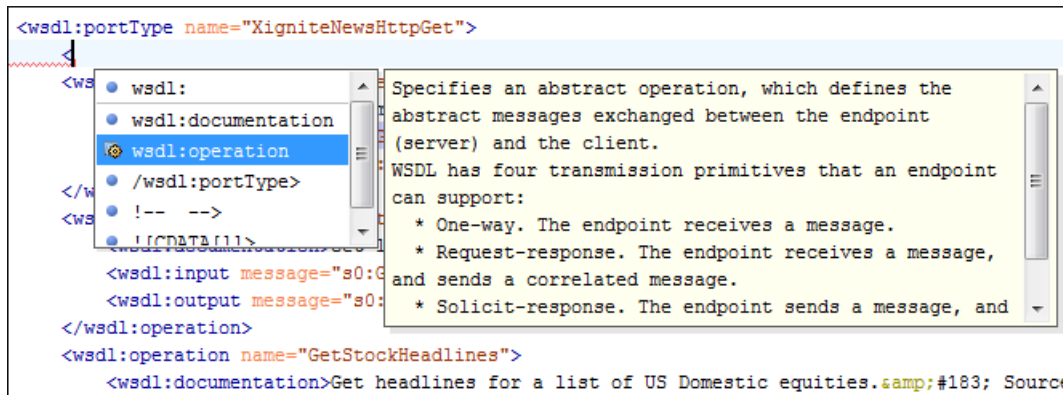



Figure 157: WSDL Content Completion Window

 **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.

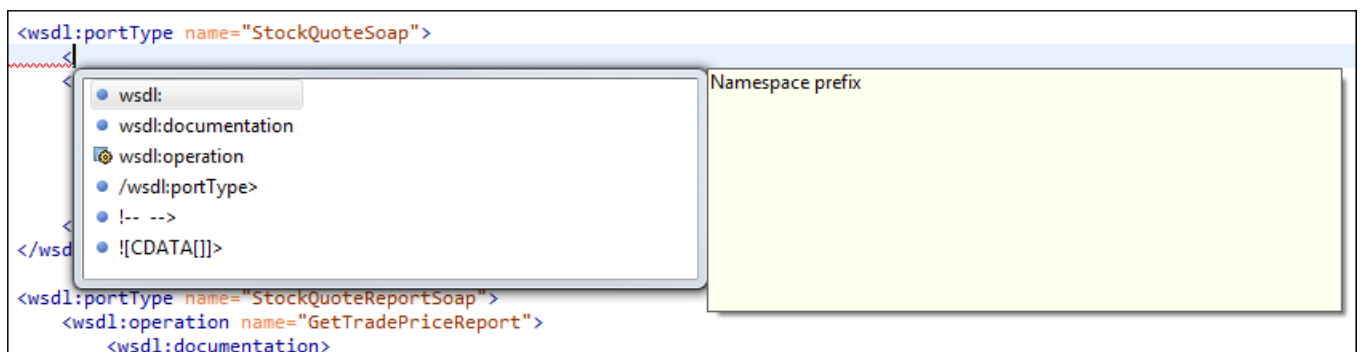


Figure 158: Namespace Prefixes in the Content Completion Window

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 plugin 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 plugin 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 Navigator 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 plugin 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://oxygenxml.com/demo/WSDL_Working_Modules.html.

Searching and Refactoring Operations in WSDL Documents

Oxygen XML Editor plugin offers support to quickly find the declaration of a component, where it is referenced, and to rename it using dedicated operations. When you rename a component, Oxygen XML Editor plugin detects all its references and updates them automatically.

The following searching and refactoring operations are available in the main toolbar of Oxygen XML Editor plugin for a WSDL document:

- **WSDL >  > Search References** - finds all the references of the component located at the caret position in the defined scope. If a scope is and the current edited resource is not part of the range of determined resources, a warning dialog is displayed. This dialog allows you to define another search scope.
- **contextual menu of current editor > Search > Search References in...** - searches for all the references of the current component. This operation demands that you define the scope of the search operation.
- **WSDL >  Search Declarations** - finds the declaration of the component located at the caret position in the defined scope. If a scope is defined and the current edited resource is not part of the range of resources determined by this scope, a warning dialog is displayed.
- **contextual menu of current editor > Search > Search Declarations in...** - searches for the declaration of the current component. This operation demands that you define the scope of the search operation.
- **WSDL > Search Occurrences in File** - searches all occurrences of the component located at the caret position in the currently edited file.



Note: The results of the search operations are presented in the **Results** view.

- **WSDL > Show Definition** - takes you to the location of the definition of the current item.



Note: You can also use the **Ctrl (Meta on Mac OS) + Click** shortcut on a reference to display its definition.

Searching and Refactoring Operations Scope in WSDL Documents


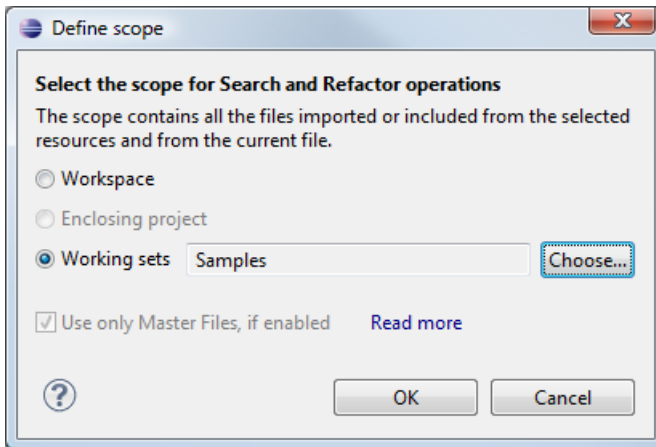
The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the  **Change scope** operation, available in the Quick Fix action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project 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 159: 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 > Other > oXygen > 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.

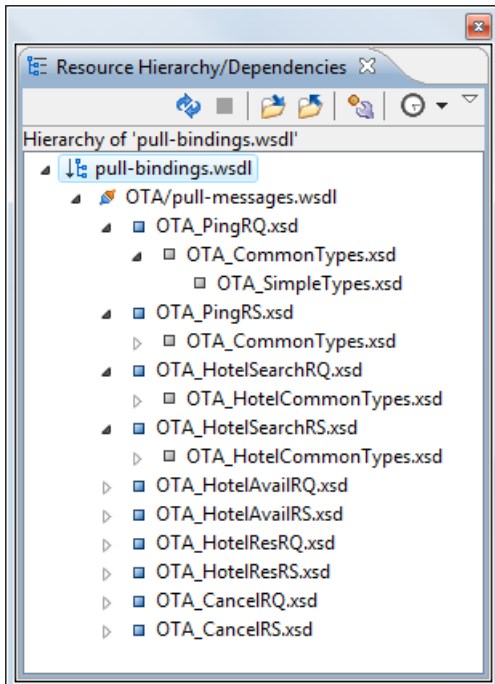


Figure 160: Resource Hierarchy/Dependencies View

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.

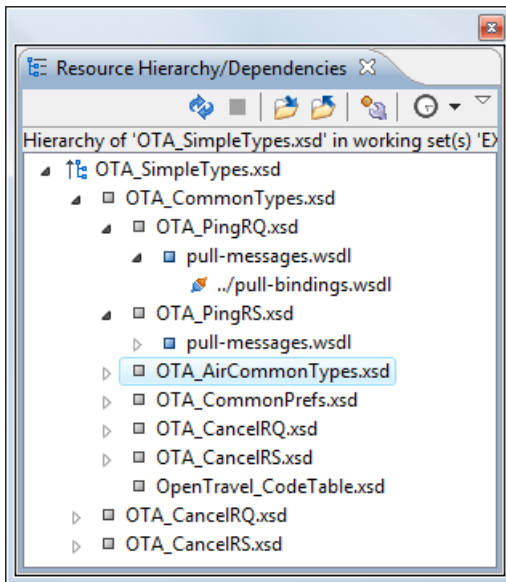


Figure 161: Resource Hierarchy/Dependencies View

The following actions are available in the **Resource Hierarchy/Dependencies** view:

- - Refreshes the Hierarchy/Dependencies structure;
- - Stop the hierarchy/dependencies computing;
- - Allows you to choose a resource to compute the hierarchy structure;
- - Allows you to choose a resource to compute the dependencies structure;
- - 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;
- - Repeats a previous dependencies computation.

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 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 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 is available for both actions. The **Preview** dialog 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 > Other > oXygen XML Editor > 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.

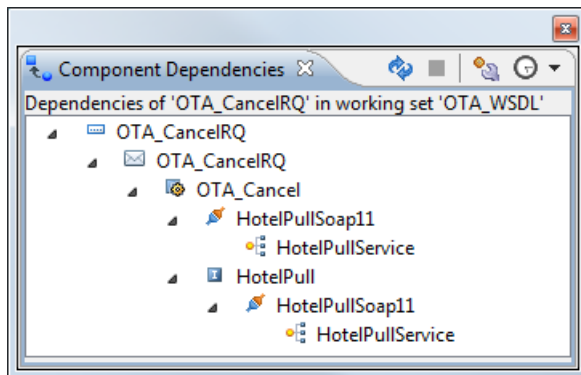






Figure 162: Component Dependencies View

The following actions are available in the toolbar of the **Component Dependencies** view:

-  - refreshes the dependencies structure.
-  - stops the dependencies computing.
-  - 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.
-  - 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 referred 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 plugin 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.

You can change the default behaviour of **Highlight Component Occurrences** from **Options > Preferences > Editor > Mark Occurrences**. You can also trigger a search using the **Search > Search Occurrences in File (Ctrl (Meta on Mac on OS)+Shift+U)** 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 **Ctrl + 1** keys (**Meta 1** on Mac OS X) on your keyboard.

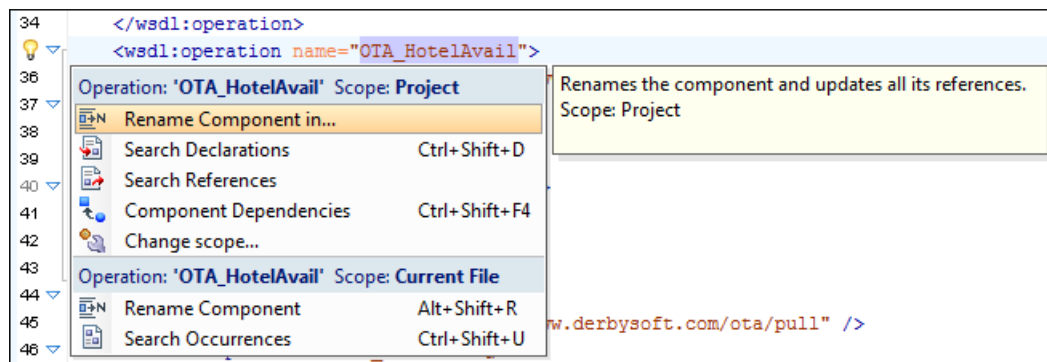


Figure 163: XSLT Quick Assist Support


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 in-place editing of the current component. When you use in-place editing, the component and all its references in the document are decorated with a thin border. All the changes you make to the component at caret position are updated in real time to all component's occurrences. To exit in-place editing, press either the **Esc** or **Enter** key on your keyboard;
- **Search Occurrences** - Searches all occurrences of the component within the current file.

Generating Documentation for WSDL Documents

You can use Oxygen XML Editor plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select what 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, go to **XML Tools > Generate Documentation > WSDL Documentation...** . You can also open the **WSDL Documentation** dialog from the contextual menu in the **Navigator: Generate WSDL Documentation**.

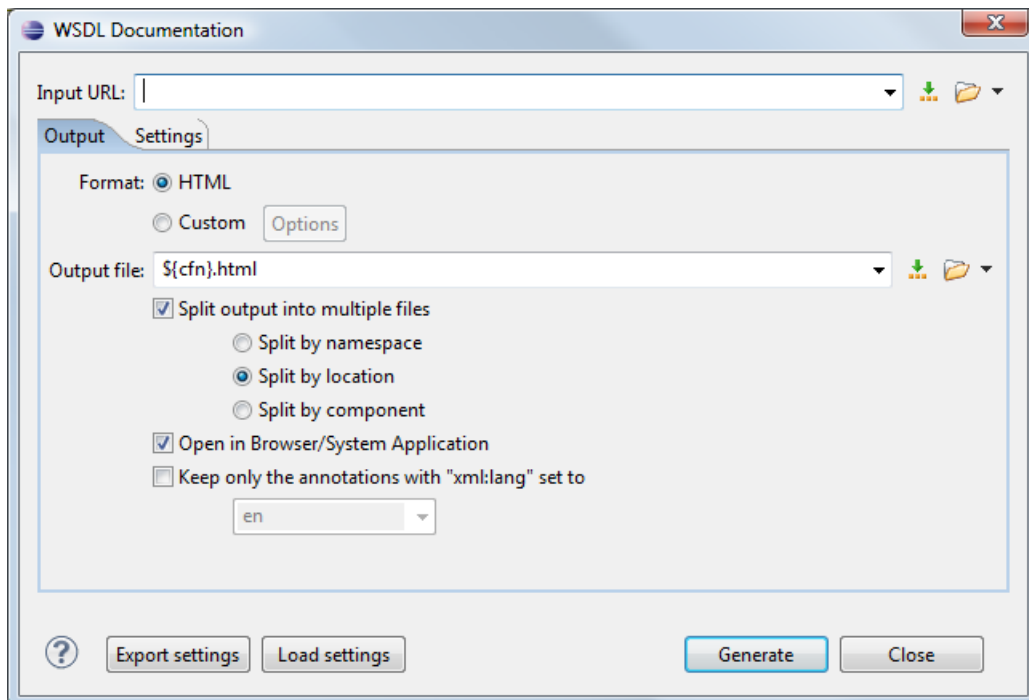


Figure 164: The Output Panel of the WSDL Documentation Dialog

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.

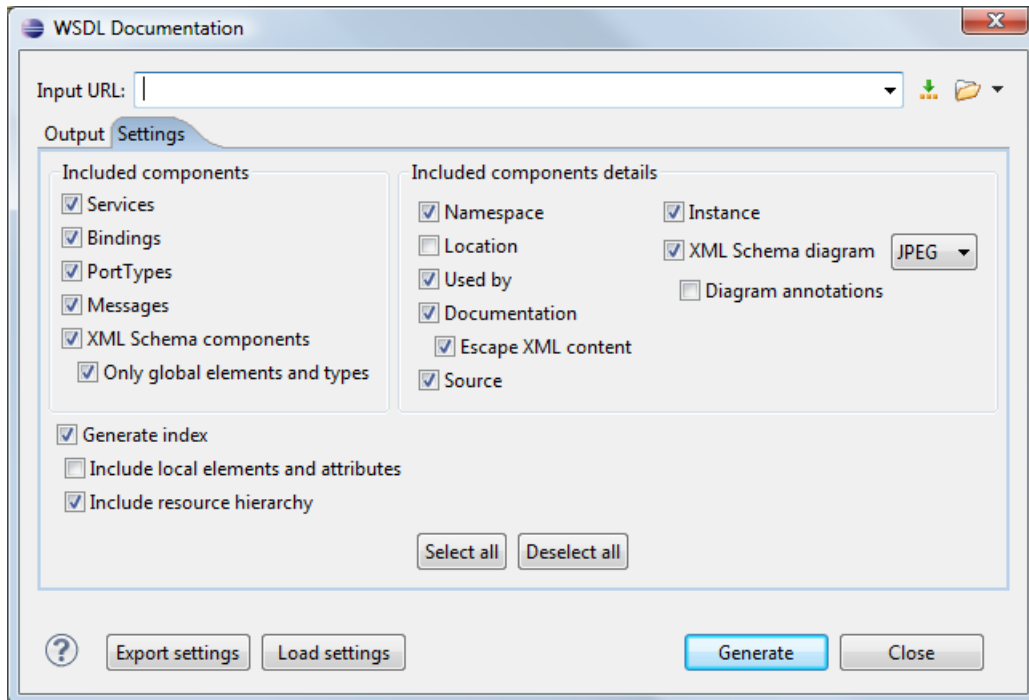


Figure 165: The Settings Panel of the WSDL Documentation Dialog


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 refer the current one;
- **Documentation** - presents the component documentation. In case you choose **Escape XML Content**, the XML tags are present 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, 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.

The screenshot displays the Oxygen XML Editor's WSDL documentation interface. On the left, the 'Table of Contents' is visible, with tabs for 'Components' and 'Resource Hierarchy'. The 'Components' tab is active, showing a tree view of the WSDL document's structure, including 'stockquoteOperations.wSDL', 'stockquote.xsd', and various messages and elements. The main content area on the right is titled 'Main WSDL stockQuoteService.wsdl' and shows details for the 'Service tns:StockQuoteService'. This section includes a table with 'Name' (StockQuote) and 'Namespace' (http://example.com/stockquote/service). Below this, the 'Service' details are shown in a tabular format, including 'Namespace', 'Documentation' (Provides the last trade price for a stock.), 'Ports' (StockQuotePort), 'Binding' (tns:StockQuoteSoap), and 'Extensibility'. The 'Source' section displays the XML code for the service definition. On the far right, the 'Showing' control panel allows users to toggle the visibility of different components: Ports, Operations, Documentation, Source, and Used by. All these options are currently checked.

Figure 166: 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.


The 'Showing' control panel is a small dialog box with a title bar 'Showing:'. It contains a list of five checkboxes, all of which are checked: 'Ports', 'Operations', 'Documentation', 'Source', and 'Used by'. A 'Close' button is located at the bottom right of the dialog.

Figure 167: 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 plugin 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 plugin, in the `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 plugin, in the `frameworks/wSDL_documentation` folder.

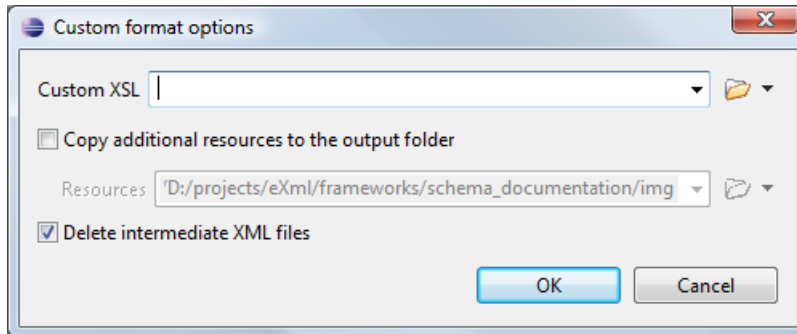


Figure 168: The Custom Format Options Dialog

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 plugin. 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 plugin's **WSDL SOAP Analyzer** 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 plugin 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 Analyzer** tool for the currently edited WSDL document do one of the following:

- click the  **WSDL SOAP Analyzer** toolbar button;

- go to **WSDL > WSDL SOAP Analyser**;
- go to in the contextual menu of the **Project** view.

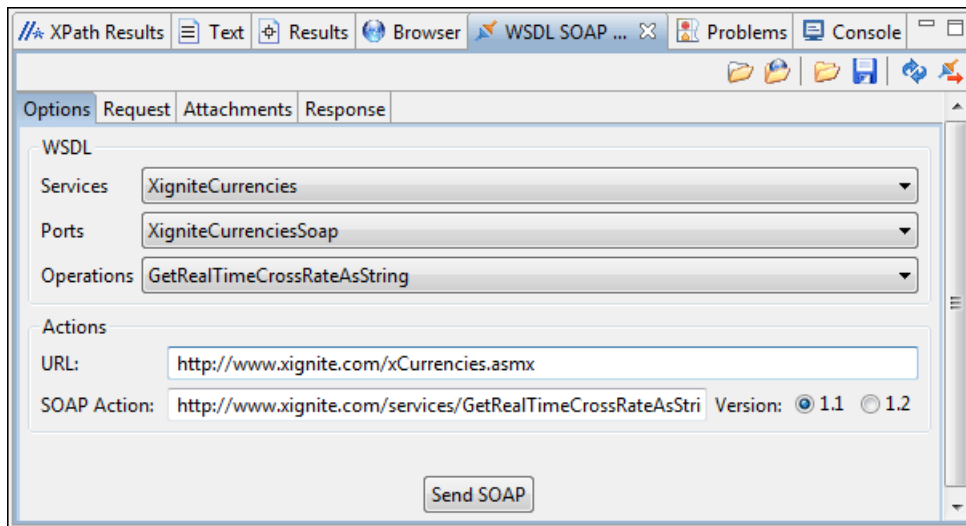



Figure 169: WSDL SOAP Analyser

This dialog 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 plugin 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 plugin 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 plugin 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 is shown when Oxygen XML Editor plugin 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 **Window > Show View > Other > oXygen > WSDL SOAP Analyser ...**
2. Press the **Choose WSDL** button and 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 Analyser**) opens the **UDDI Registry Browser** dialog.

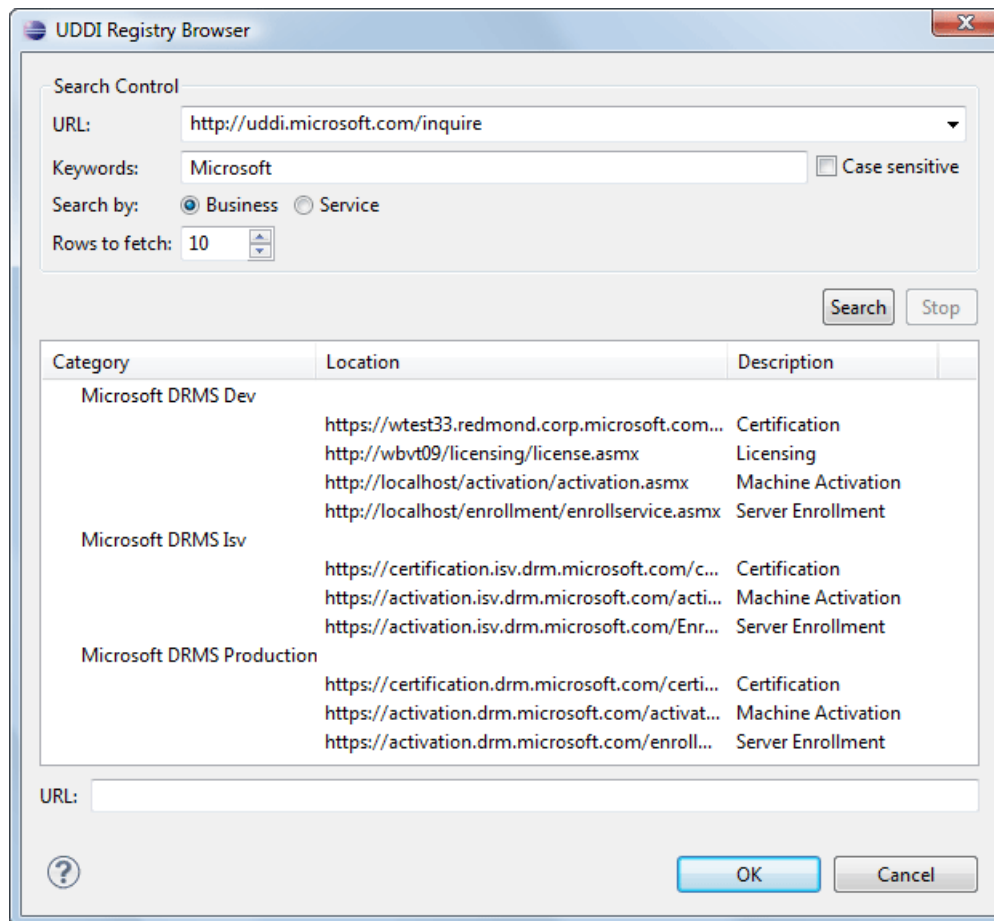


Figure 170: UDDI Registry Browser dialog

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 plugin includes a built-in CSS validator integrated with the general validation support. This makes the *usual validation features* also available for CSS stylesheets.

The CSS properties accepted by the validator are the ones 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 and 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

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 your desktop. From its 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-install-dir]/lib` and create the `endorsed` folder. Copy the `custom_props.jar` file to `[Oxygen-install-dir]/lib/endorsed`.

Content Completion in CSS Stylesheets

A **Content Completion Assistant** like *the one available for XML documents* offers the CSS properties and the values available for each property. It is activated on the **Ctrl (Meta on Mac OS) + Space** shortcut and it is context-sensitive when invoked for the value of a property.

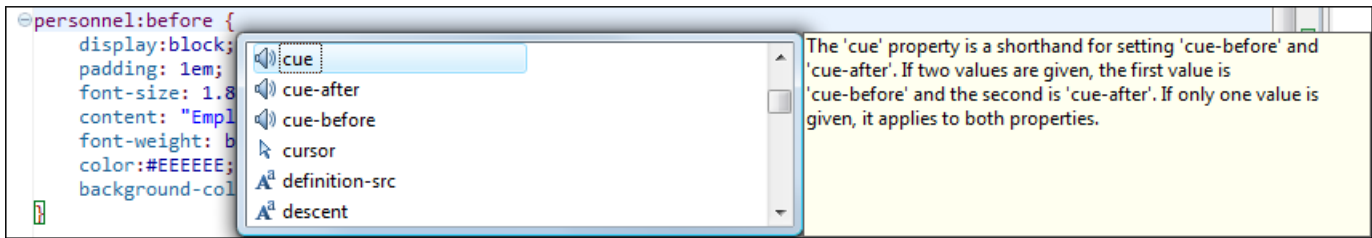


Figure 171: Content Completion in CSS Stylesheets

The properties and the values offered as proposals are dependent on the CSS Profile selected in the [Options > Preferences > CSS Validator](#) page, **Profile** combo box. The CSS 2.1 set of properties and property values is used for most of the profiles, excepting CSS 1 and CSS 3. For these two, specific proposal sets are used.

The profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties and 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 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 the changes you make in the stylesheet document. When you select an entry from the **Outline** view, Oxygen XML Editor plugin highlights the corresponding import or selector in the CSS editor.

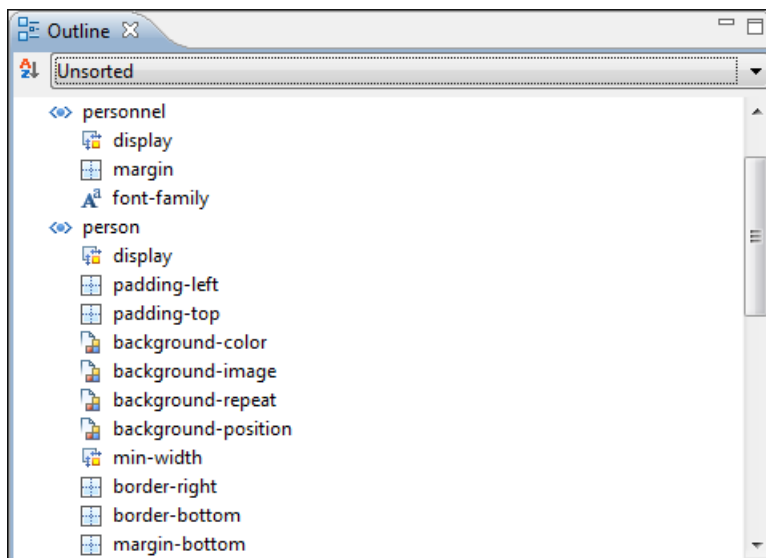


Figure 172: CSS Outline View

The selectors presented in this view can be quickly found 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 needed styles 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 after an opening or in front of a closing bracket.

Formatting and Indenting CSS Stylesheets (Pretty Print)

If the edited CSS stylesheet becomes unreadable because of the bad alignment of the text lines, *the pretty-print 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.

Other CSS Editing Actions

The CSS editor type offers a reduced version of *the popup menu available in the XML editor*. Only *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 Relax NG Schemas

Oxygen XML Editor plugin 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 plugin 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 Navigator 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 plugin 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 plugin 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 plugin 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**.

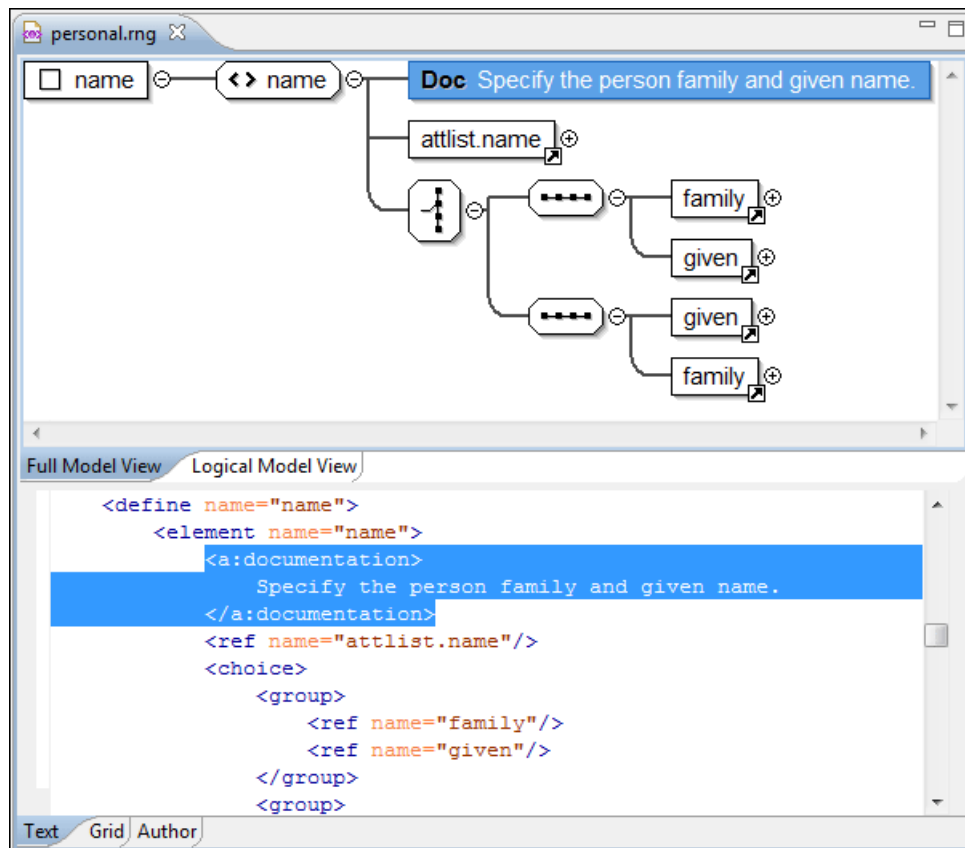


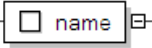


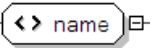
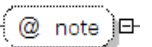
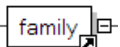




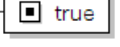

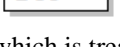


Figure 173: Relax NG Schema Editor - Full Model View

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.

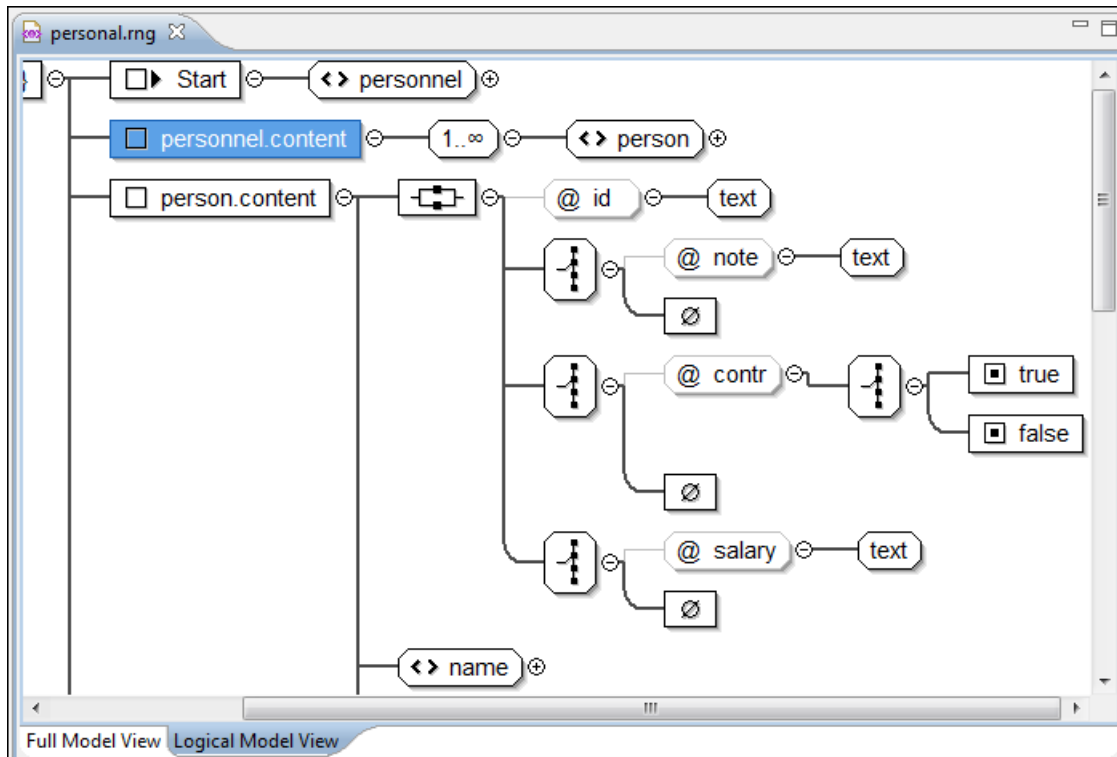


Figure 174: Logical Model View for a Relax NG Schema

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 referred 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 > Other > oXygen > Outline**.

You can switch between the Relax NG patterns version and the *standard XML version* of the view by pressing the  button.

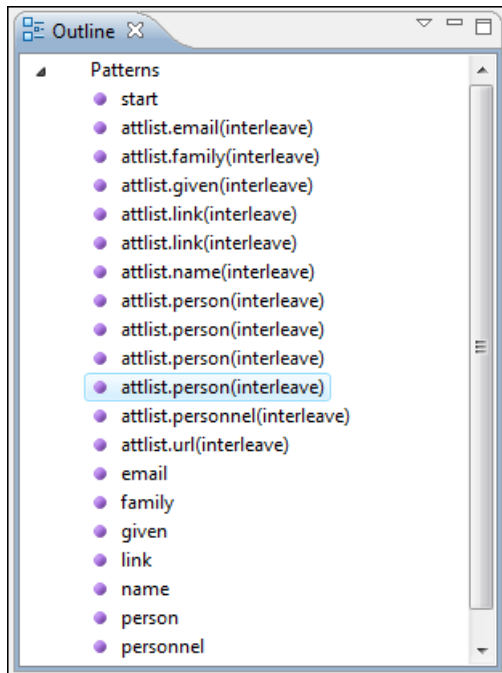











Figure 175: Relax NG Outline View

The tree shows the XML structure or the define patterns collected from the current document. By default, the **Outline** view presents the define patterns. The following action is available in the **View menu** on the Outline view's action bar:

-  **Show XML structure** - Shows the XML structure of the current document.

When the XML elements are displayed, the following actions are available in the **View menu** on the Outline view's action bar:

-  **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](#).

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:

- **contextual menu of current editor > Show Definition** - Moves the cursor to the definition of the current element in this Relax NG (full syntax) schema. You can use the **Ctrl (Meta on Mac OS) + Click** shortcut on a reference to display its definition.

Searching and Refactoring Actions

All the following actions can be applied on `ref` and `parentRef` parameters only.

- **RNG > 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 scope, a warning dialog is shown and you can define another search scope. A search scope includes the project or a collection of files and folders.

You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

- **contextual menu of current editor > Search > Search References in...** - Searches all references of the item found at current cursor position in the file or files specified in the defined scope.

All the following actions can be applied on named `define` parameters only.

- **RNG > 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 is shown and you can define another search scope. A search scope includes the project or a collection of files and folders.
- **contextual menu of current editor > Search > Search Declarations in...** - Searches all declarations of the item found at current cursor position in the files specified in the search scope.
- **RNG > Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor > Refactoring > Rename Component...** - Renames the selected component.

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 > Other > oXygen > 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.

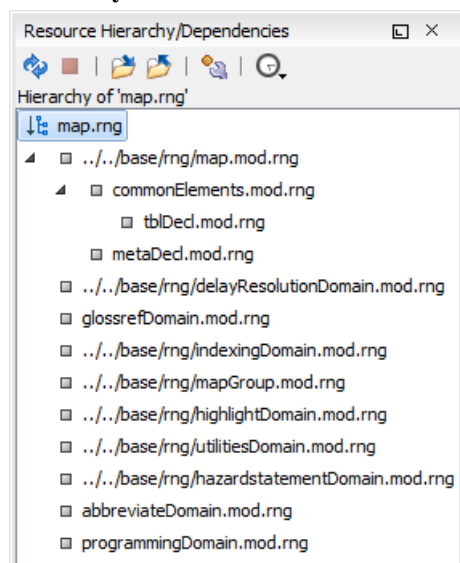


Figure 176: Resource Hierarchy/Dependencies View - hierarchy for `map.rng`

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.

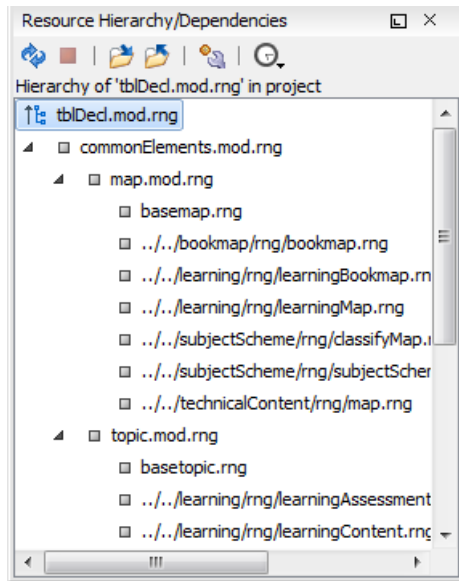









Figure 177: Resource Hierarchy/Dependencies View - Dependencies for tblDed1.mod.rng


The following actions are available in the **Resource Hierarchy/Dependencies** view:

-  - Refreshes the Hierarchy/Dependencies structure;
-  - Stop the hierarchy/dependencies computing;
-  - Allows you to choose a resource to compute the hierarchy structure;
-  - Allows you to choose a resource to compute the dependencies structure;
-  - 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;
-  - Repeats a previous dependencies computation.

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 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 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 is available for both actions. The **Preview** dialog 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 > Other > oXygen XML Editor > 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.

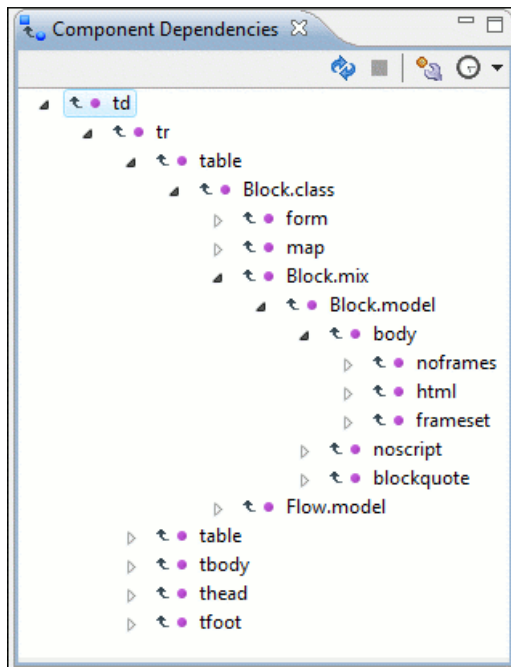








Figure 178: Component Dependencies View - Hierarchy for xhtml1.rng

In the **Component Dependencies** view you have several actions in the toolbar:

-  - Refreshes the dependencies structure.
-  - Allows you to stop the dependencies computing.
-  - Allows you to configure a search scope to compute the dependencies structure.
-  - 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 referred 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 Fix support improves the development work flow, offering fast access to the most commonly used actions when you edit 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 **Ctrl + 1** keys (**Meta 1** on Mac OS X) on your keyboard.

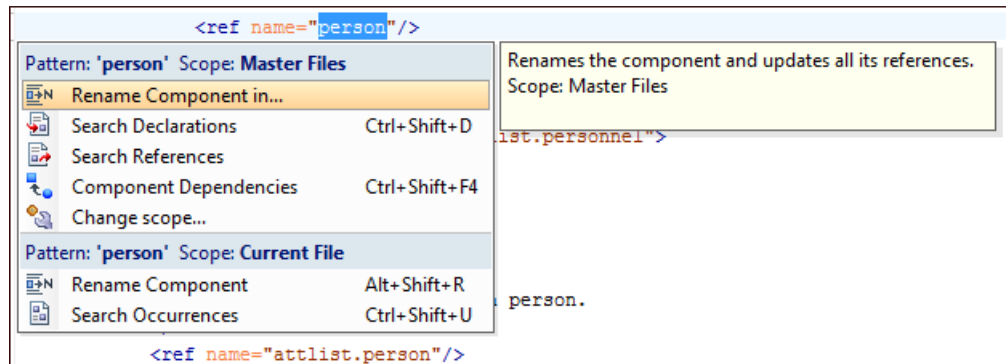


Figure 179: RNG Quick Assist Support

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 in-place editing of the current component. When you use in-place editing, the component and all its references in the document are decorated with a thin border. All the changes you make to the component at caret position are updated in real time to all component's occurrences. To exit in-place editing, press either the **Esc** or **Enter** key on your keyboard;
- **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).

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-plugin-folder]/lib` and a line `<library name="lib/custom-library.jar"/>` must be added for each jar file to the file `[Oxygen-plugin-folder]/plugin.xml` in the `<runtime>` element.

To load the custom library, restart the Eclipse platform.

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 plugin 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 plugin 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 plugin 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**.

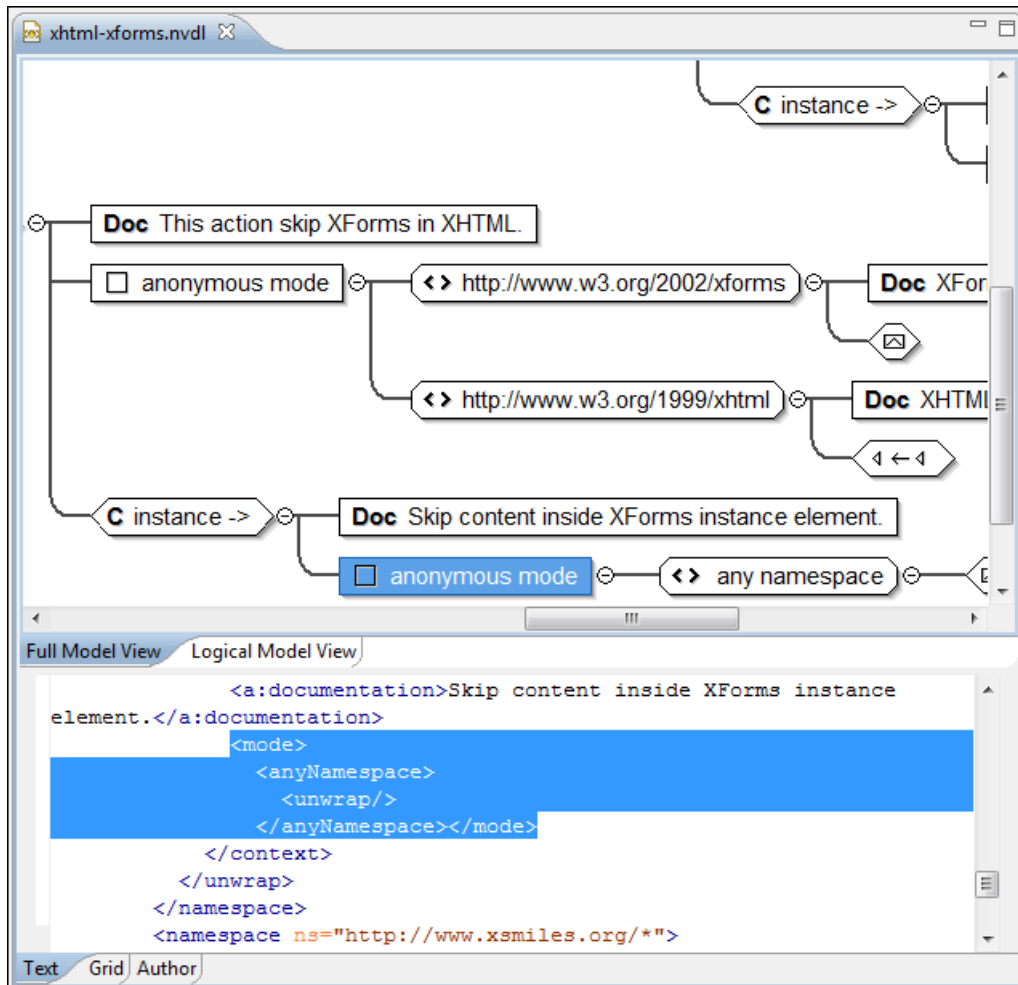


Figure 180: NVDL Schema Editor - Full Model View

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 referred 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 > Other > oXygen XML Editor > Outline** menu.


NVDL Editor Specific Actions

The list of actions specific for the Oxygen XML Editor plugin NVDL editor of is:


- **contextual menu of current editor > Show Definition** - Moves the cursor to its definition in the schema used by NVDL in order to validate it. You can use the **Ctrl (Meta on Mac OS) + Click** shortcut on a reference to display its definition.

Searching and Refactoring Actions

You can apply the following actions on mode name, useMode, and startMode attributes only:

- **NVDL >  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 scope, a warning dialog is shown. A search scope includes the project or a collection of files and directories. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.
- **contextual menu of current editor > Search > 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 dialog above.

All the following actions can be applied on named `define` parameters only.

- **NVDL >  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 is shown. You have the possibility to define another search scope. A search scope includes the project or a collection of files and folders.
- **contextual menu of current editor > Search > Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files specified in the search scope.
- **NVDL > Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor > Rename Component...** - Allows you to rename the current component.

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 > Other > <oXygen/> XML > 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.

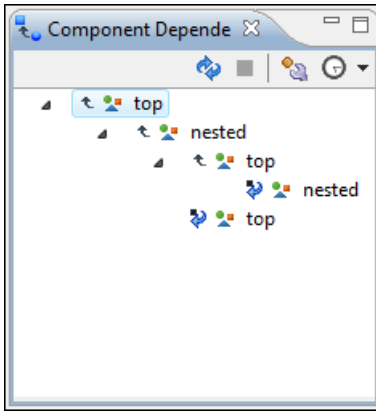








Figure 181: Component Dependencies View - Hierarchy for test.nvdl

In the **Component Dependencies** the following actions are available on the toolbar:

-  - Refreshes the dependencies structure.
-  - Allows you to stop the dependencies computing.
-  - 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.
-  - Repeats a previous dependencies computation.

The following actions are available in the contextual menu:

- **Go to First Reference** - Selects the first reference of the referred 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 plugin 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**.

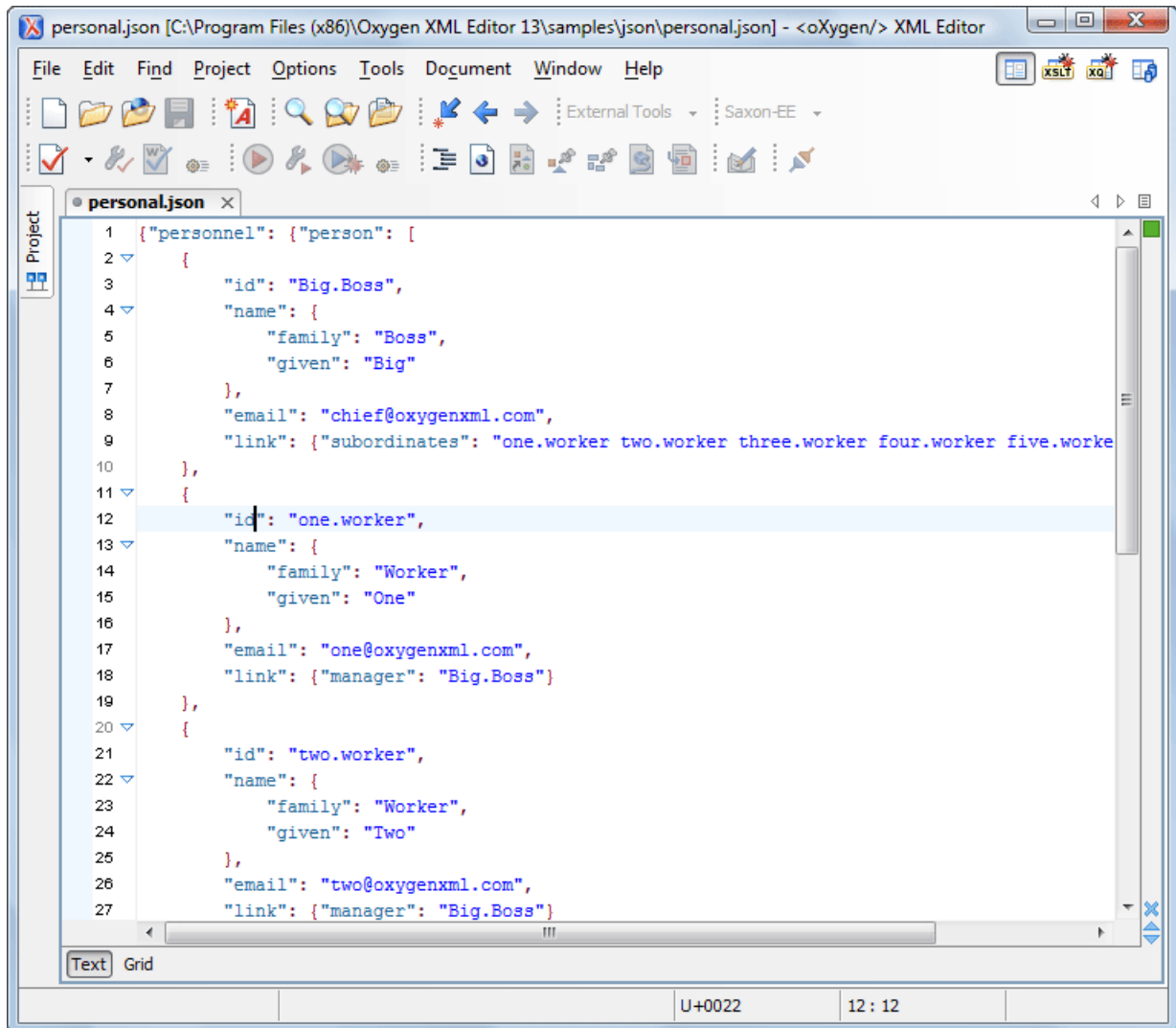


Figure 182: JSON Editor Text Mode

Syntax highlight in JSON Documents

Oxygen XML Editor plugin 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

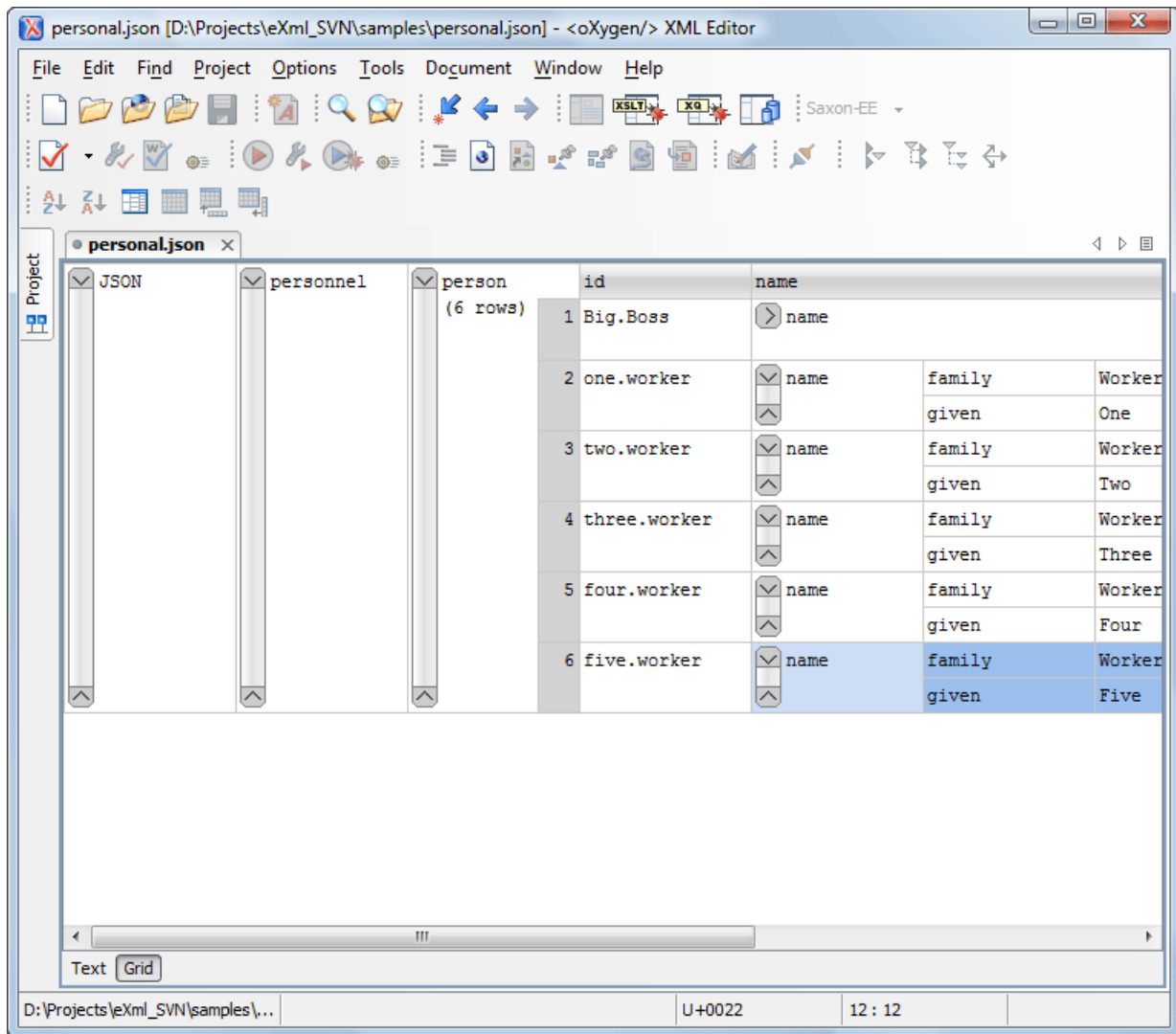


Figure 183: JSON Editor Grid Mode

Oxygen XML Editor plugin 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**.

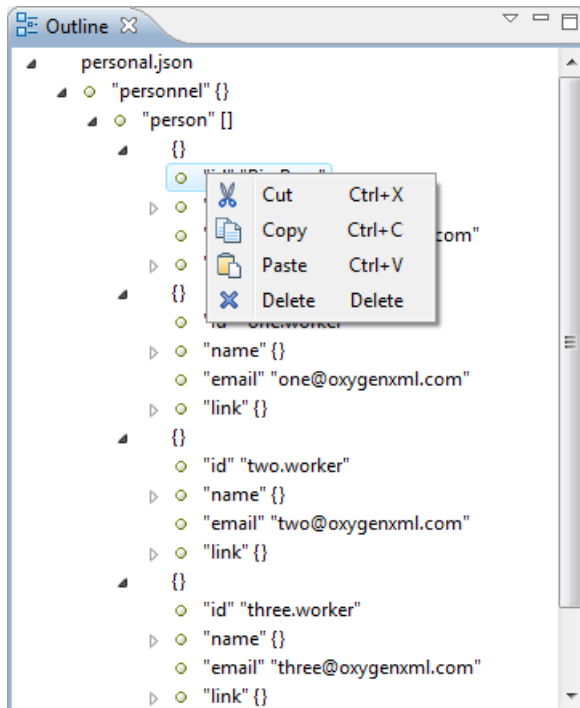







Figure 184: The JSON Outline View

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 plugin 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.

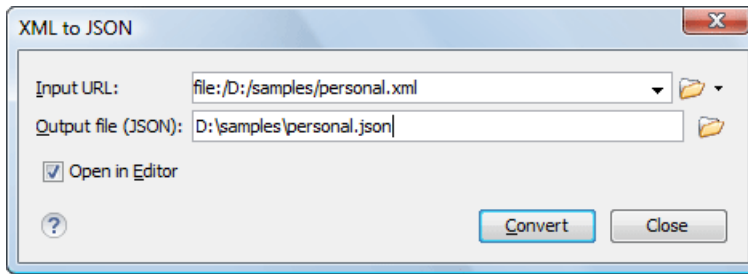
Validating JSON Documents

Oxygen XML Editor plugin 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

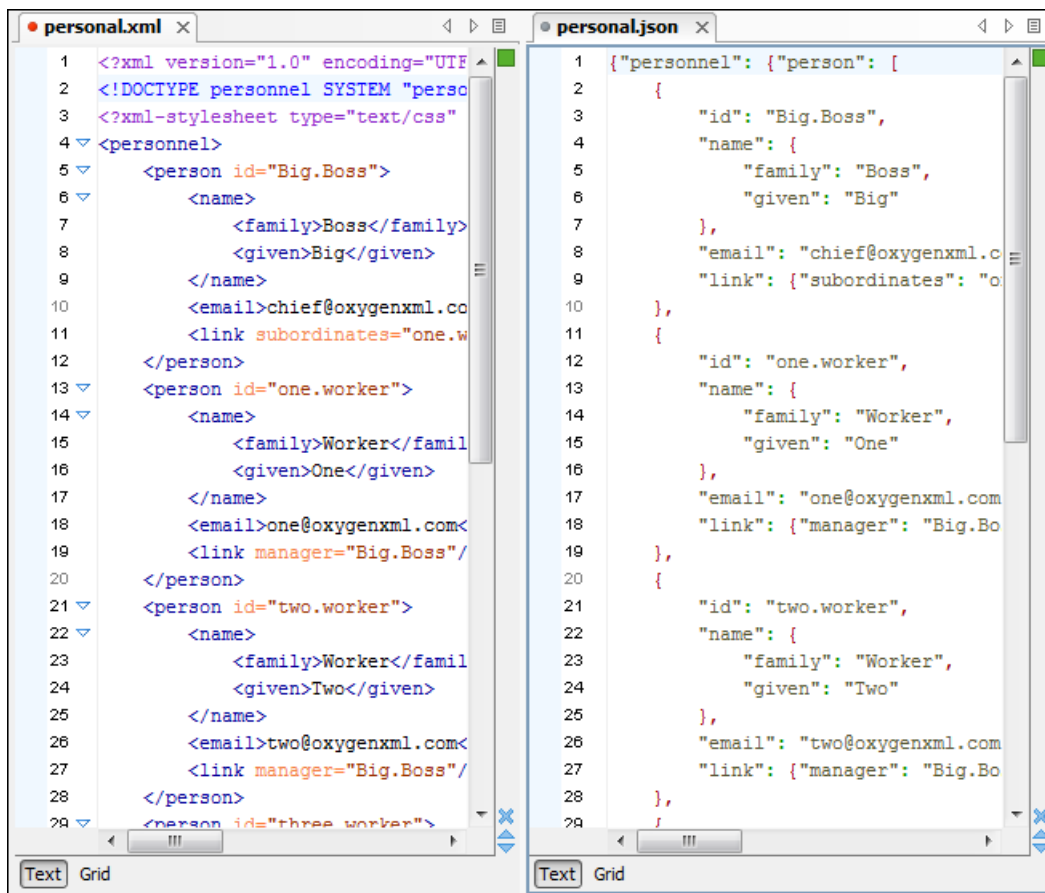
The steps for converting an XML document to JSON are the following:

1. Go to menu **XML Tools > XML to JSON...**
The **XML to JSON** dialog 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 plugin JSON Editor
5. Click the **OK** button.

The operation result will be a document containing the JSON conversion of the input XML URL.



Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Editor plugin 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 plugin 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 plugin JavaScript Editor and how you can use them.

JavaScript Editor Text Mode

Oxygen XML Editor plugin 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 after an opening or in front of a closing bracket.






```

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 185: JavaScript Editor Text Mode

The contextual menu of the **JavaScript** editor offers the following options:

-  **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 plugin 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 options:
 - **Open File at Caret** - select this option to open the source of the file located at the caret position;
 - **Open File at Caret in System Application** - select this option to open the source of the file located at the caret position with the application that the system associates with the file;
 - **Open in Browser/System Application** - select this option to open the file in the system application associated with the file type.








Note: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes precedence over the default system application settings.

- **Folding** - allows you to select one of the following options:
 -  **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 options:
 - **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** - 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 =
20                 var mytoc =
21                 if (mytoc)
22                 mytoc.lastU
23                 }
24                 }
25                 }
26                 }
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 186: 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 plugin 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**.

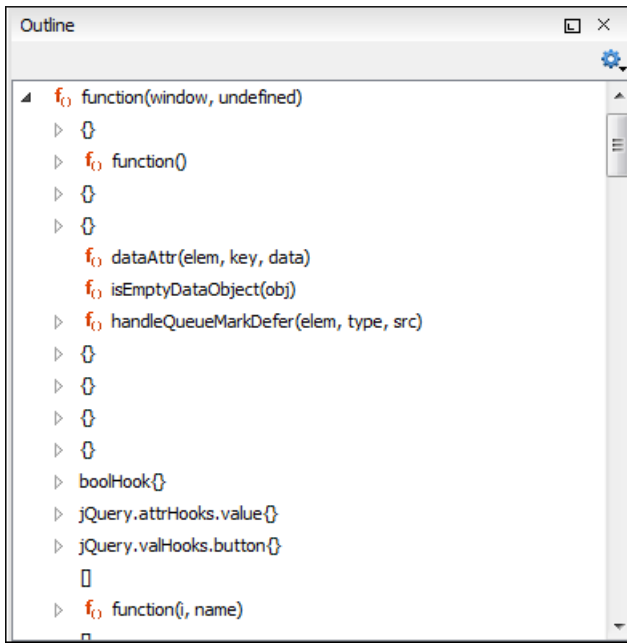











Figure 187: The JavaScript Outline View

The following icons decorate the elements in the **Outline** view 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.

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 plugin 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 plugin 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 with Oxygen XML Editor plugin version 15.2.

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 on the `port` attribute of `input` and the parent of `input`. When invoking the content completion inside the XProc element `inline`, depending on the attribute `port` of its parent `input` element and the parent of element `input`, elements from different schemas are offered inside the proposals list.

- 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 renders with dedicated coloring schemes the XPath expressions. You can customize the coloring schemes in the **Window > Preferences > oXygen XML Editor > Editor > Syntax Highlight** preferences page.

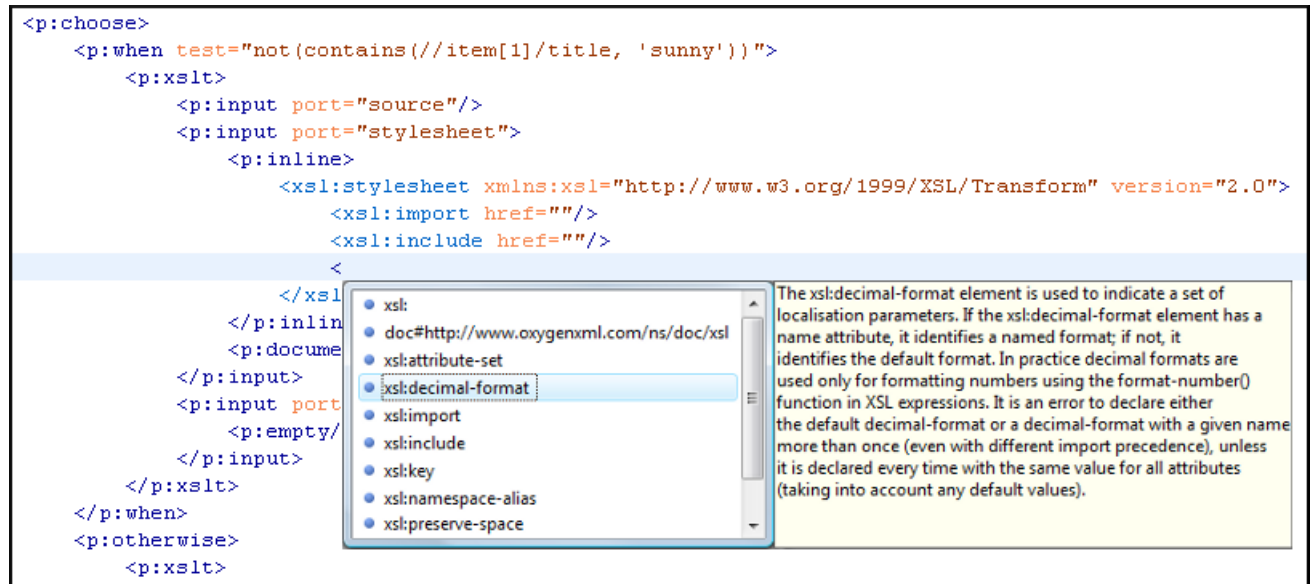


Figure 188: XProc Content Completion

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 plugin 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, go to **Window > Preferences > oXygen XML Editor > Editor > Syntax Highlight**.



Note: When you create a Schematron document, Oxygen XML Editor plugin 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, invoke the **Validate** action either from the application's toolbar or from the **Project** view's contextual menu. If you would like to add a persistence association between your Schematron rules and the current edited XML document, use the Associate Schema action. 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 plugin 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 plugin 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 plugin 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 plugin sets the severity level to *error*.

Validating Schematron Documents

To validate your Schematron document, use the **Validate** action from the main toolbar. When Oxygen XML Editor plugin 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 plugin offers an error management mechanism capable of pinpointing errors in XPath expressions and in the included schema modules.



Note: By default, a Schematron schema is validated as you type. You can change this from the **Option > Preferences > Editor > Document Checking** preferences page.

Content Completion in Schematron Documents

Oxygen XML Editor plugin 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.

Code Templates

When the content completion is invoked by pressing **(CTRL (Meta on Mac OS)+Space)**, it also presents a list of code templates specific to the type of the active editor. Such a code template provides a shortcut for inserting a small document fragment at the caret position. Oxygen XML Editor plugin comes with a set of ready-to use templates for Schematron documents.

The Schematron code template called Pattern-Rule-Assert

Typing `p` in a Schematron document and selecting `pra` in the content assistant pop-up window inserts the following template at the caret position in the document:

```
<pattern id="">
  <rule context="">
    <!-- Write your assertions or reports here -->
    <assert test=""></assert>
  </rule>
</pattern>
```

You *can easily define other templates* and share them with other users.

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 plugin 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 plugin 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 plugin 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 Navigator 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 plugin 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 > Other > oXygen > 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.

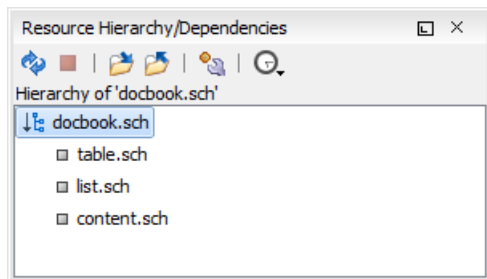


Figure 189: Resource Hierarchy/Dependencies View

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.

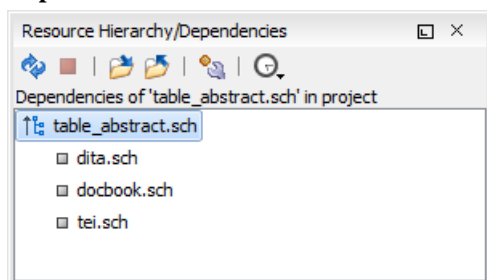









Figure 190: Resource Hierarchy/Dependencies View - Dependencies for table_abstract.sch



The following actions are available in the **Resource Hierarchy/Dependencies** view:


-  - Refreshes the Hierarchy/Dependencies structure;
-  - Stop the hierarchy/dependencies computing;

-  - Allows you to choose a resource to compute the hierarchy structure;
-  - Allows you to choose a resource to compute the dependencies structure;
-  - 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;
-  - Repeats a previous dependencies computation.

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 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 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 is available for both actions. The **Preview** dialog 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 plugin 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.

You can change the default behaviour of **Highlight Component Occurrences** from **Options > Preferences > Editor > Mark Occurrences**. You can also trigger a search using the **Search > Search Occurrences in File (Ctrl (Meta on Mac on OS)+Shift+U)** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in Schematron Documents

Oxygen XML Editor plugin offers support to quickly find the declaration of a component, where it is referenced, and to rename it using dedicated operations. When you rename a component, Oxygen XML Editor plugin detects all its references and updates them automatically.


The following searching and refactoring operations are available in the main toolbar of Oxygen XML Editor plugin for a Schematron document:

- **Contextual menu of current editor > Search >  > Search References** - finds all the references of the component located at the caret position in the defined scope. If a scope is and the current edited resource is not part of the range of determined resources, a warning dialog is displayed. This dialog allows you to define another search scope.
- **Contextual menu of current editor > Search > Search References in...** - searches for all the references of the current component. This operation demands that you define the scope of the search operation.
- **Contextual menu of current editor > Search >  Search Declarations** - finds the declaration of the component located at the caret position in the defined scope. If a scope is defined and the current edited resource is not part of the range of resources determined by this scope, a warning dialog is displayed.
- **Contextual menu of current editor > Search > Search Declarations in...** - searches for the declaration of the current component. This operation demands that you define the scope of the search operation.
- **Contextual menu of current editor > Search > Search Occurrences in File** - searches all occurrences of the component located at the caret position in the currently edited file.



Note: The results of the search operations are presented in the **Results** view.

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 Fix 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](#).

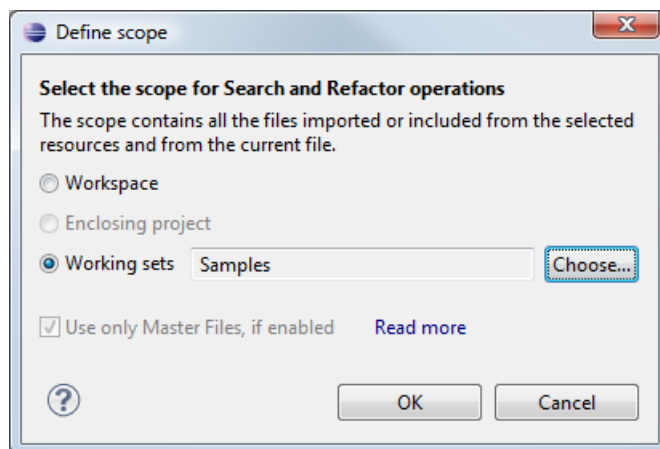


Figure 191: Define Scope Dialog

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 **Ctrl + 1** keys (**Meta 1** on Mac OS X) on your keyboard.

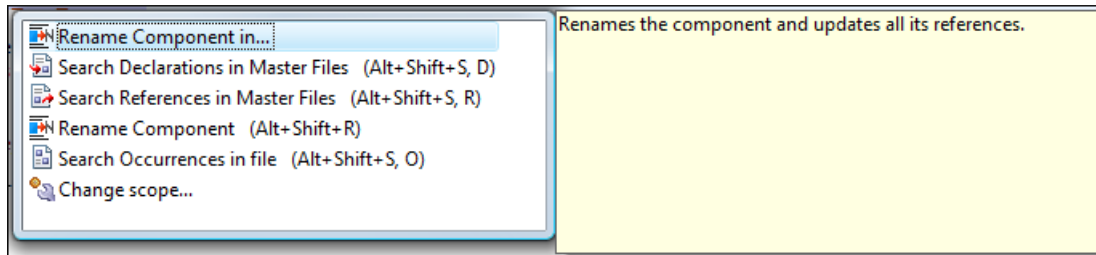



Figure 192: Schematron Quick Assist Support

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 in-place editing of the current component. When you use in-place editing, the component and all its references in the document are decorated with a thin border. All the changes you make to the component at caret position are updated in real time to all component's occurrences. To exit in-place editing, press either the **Esc** or **Enter** key on your keyboard;
- **Search Occurrences** - Searches all occurrences of the component within the current file.

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.

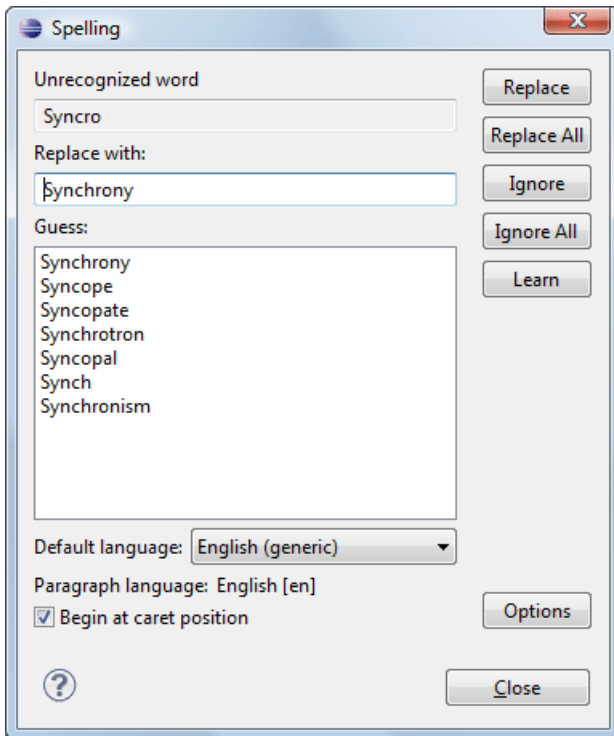


Figure 193: The Check Spelling Dialog

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 plugin 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 plugin 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 plugin: **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 plugin.

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 plugin 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*.

Adding a Dictionary and Term Lists for the Hunspell Checker

To add a new spelling dictionary to Oxygen XML Editor plugin 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. Copy the .aff and .dic files from this archive in the spell subfolder of the Oxygen XML Editor plugin preferences folder, if you are creating a new dictionary.

The Oxygen XML Editor plugin preferences folder is >, where [APPLICATION-DATA-FOLDER] is:

- C:\Users\[LOGIN-USER-NAME]\AppData\Roaming on Windows Vista and Windows 7;
- [USER-HOME-FOLDER]/Library/Preferences on Mac OS X;
- [USER-HOME-FOLDER] on Linux.

3. Copy the .aff and .dic files into the folder [Oxygen-install-folder]/dicts if you are updating an existing dictionary.
4. Restart the application after copying the dictionary files.



Note: You can setup Oxygen XML Editor plugin 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-install-folder]/dicts. Oxygen XML Editor plugin 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-install-folder]/dicts` and restarting Oxygen XML Editor plugin. 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 so:

- press the **Learn** button from the **Spelling** dialog;
- 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:

- `[user-home-folder]/Application Data/com.oxygenxml/spell` folder on Windows XP;
- `[user-home-folder]/AppData/Roaming/com.oxygenxml/spell` folder on Windows Vista;
- `[user-home-folder]/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 plugin 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 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.

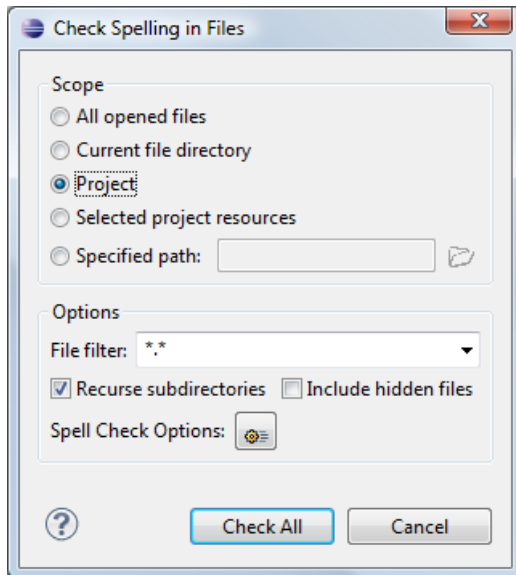


Figure 194: Check Spelling in Files Dialog

The following scopes are available:

- **All opened files** - spell check in all opened files;
- **Directory of the current file** - all the files from 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.

When you invoke the **Check Spelling in Files** action in the **DITA Maps Manager** view, a different dialog is displayed:

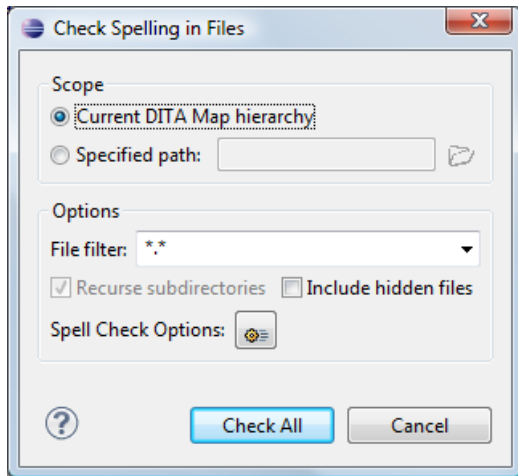


Figure 195: Check Spelling in Files Dialog in The DITA Maps Manager View

The following scopes are available:

- **Current DITA Map hierarchy** - All the files referred in the current DITA map opened in the **DITA Maps Manager** view;
- **Specified path** - checks the spelling in the files located at a path that you specify;

You can also choose a file filter, decide whether to recurse subdirectories or process hidden files.

The spell checker processor uses the options available in the [Spell Check preferences panel](#).

Handling Read-Only Files

The default workbench behavior applies when editing read-only files in the **Text** mode. For all other modes no modification is allowed as long as the file remains read-only.

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.

Associating a File Extension with Oxygen XML Editor plugin

To associate a file extension with Oxygen XML Editor plugin 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 plugin, then click **Change program**;
- in the **Open With** dialog click **Browse** and navigate to Oxygen XML Editor plugin.

To associate a file extension with Oxygen XML Editor plugin 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 plugin;
- With Oxygen XML Editor plugin selected, click **Change All**.

Terms

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	A block is an element that takes up the entire available width. A block is delimited by line breaks before and after it.
Inline	An inline element takes up as much width as necessary, and does not force line breaks.
Inline XML element	Inline elements are elements which appear in a mixed-content context (parents with both non-whitespace text and elements).

Chapter

6

Author for DITA

Topics:

- [Creating DITA Maps and Topics](#)
- [Editing DITA Maps](#)
- [Transforming DITA Maps and Topics](#)
- [DITA-OT Customization](#)
- [DITA Specialization Support](#)
- [Use a New DITA Open Toolkit in Oxygen XML Editor plugin](#)
- [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 the following topic types:

- *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;
- *Reference* - For reference information.

You can organize topics into a DITA map or bookmap. A map is a hierarchy of topics. A bookmap supports also book divisions such as chapters and book lists such as indexes. Maps do not contain the content of topics, but only references to them. These are known as topic references. Usually the maps and bookmaps 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.

Editing DITA Maps

Oxygen XML Editor plugin provides a special view for editing DITA Maps. The **DITA Maps Manager** view presents a DITA Map in a simplified table-of-contents manner. It allows you to navigate easily to the referred topics and maps, make changes, and apply transformation scenarios to obtain various output formats. The DITA-OT framework that is bundled with Oxygen XML Editor plugin is used.

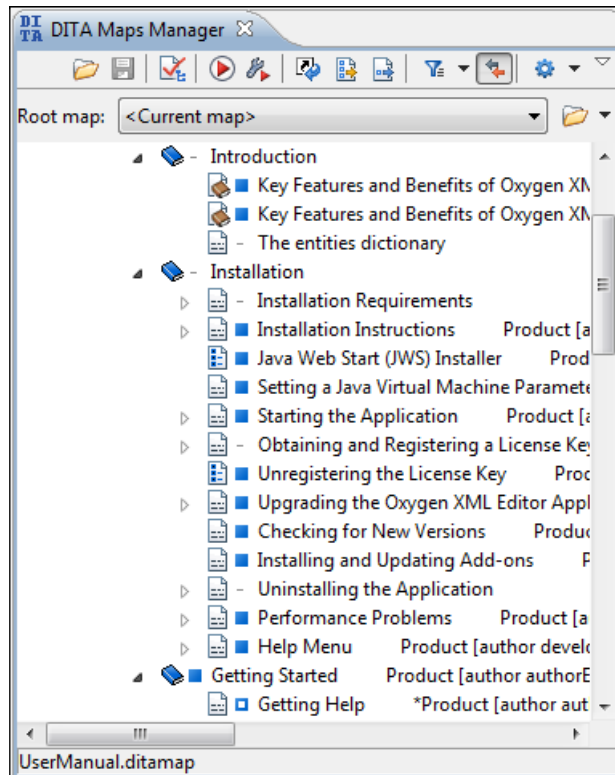












Figure 196: The DITA Maps Manager View


To open a DITA Map from the **Project** view in the **DITA Maps Manager** view, right click it and select **Open in DITA Maps Manager**. The titles of the referenced resources are resolved and displayed in the view dynamically when navigating the tree. After the DITA Map is opened, you can open it in the main editor for further customization using the **Open Map in Editor** toolbar action.


You can move topics in the same map or between different maps by dragging and dropping them into the desired position. Also, you can copy topics by dragging them while pressing the **Ctrl (Meta on Mac OS)** key.



The toolbar includes the actions which are also available on menu **DITA Maps**:

-  **Open** - Allows opening the map in the **DITA Maps Manager** view. You can also open a map by dragging it in the **DITA Maps Manager** view from the file system explorer;
-  **Open URL** - Allows opening remote maps in the **DITA Maps Manager** view. See [Open URL](#) for details;
- **Open from Data Source Explorer** - Allows you to open a DITA Map from a data source explorer;
-  **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 from the view;
-  **Configure Transformation Scenario(s)...** - Allows *associating 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 titles of referred topics. This action is useful when the referred topics are modified externally. When they are modified and saved from the Oxygen XML Editor plugin 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.
-  **Open Map in Editor** - For complex operations which cannot be performed in the simplified DITA Maps view (like 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**.

-  **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.











-  **Profiling/Conditional Text** menu with the following actions:
 - **Show Profiling Attributes** - Enables/Disables displaying 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 both in the **DITA Maps Manager** view and in the **Author** editor;
 -  **Configure Profiling Condition Sets ...** - Opens the preferences page for adding and editing the profiling conditions that you can apply in the **DITA Maps Manager** view and the **Author** editor;

If your map references other DITA Maps they will be shown expanded in the DITA Maps tree and you will also be able to navigate their content. For editing you will have 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 referred content** checkbox available in the [Author preferences page](#).

 **Tip:** The additional edit toolbar can be shown by clicking the "Show/Hide additional toolbar" expand button located on the general toolbar.

The following edit actions can be performed on an opened DITA Map:


-  **Insert Reference** - Inserts a reference to a topic file. You can find more details about this action in the [Inserting a Reference, a Key Definition, a Topic Set](#) topic;

-  **Insert Topic Heading** - Inserts a topic heading. You can find more details about this action in the *Inserting a Topic Heading* topic;
 -  **Insert Topic Group** - Inserts a topic group. You can find more details about this action in the *Inserting a Topic Group* on page 307 topic;
 -  **Edit Properties** - Edit the properties of a selected node. You can find more details about this action in the *Edit Properties* on page 308 topic;
 - **Edit Profiling Attributes** - Allows you to select the profiling attributes;
 -  **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 68 topic;
 - **Rename resource** - allows you to *change the name of a resource linked in the edited DITA Map*;
 - **Move resource** - allow you to *change the location on disk of a resource linked in the edited DITA Map*;
 -  **Delete** - Deletes the selected node;
 -  **Move Up (Alt + Up Arrow)** - Moves the selected nodes in front of their respective previous siblings;
 -  **Move Down (Alt + Down Arrow)** - Moves the selected nodes after their next respective siblings;
 -  **Promote (Alt + Left Arrow)** - Moves the selected nodes after their respective parents as siblings;
 -  **Demote (Alt + Right Arrow)** - Moves the selected nodes as children to their respective previous siblings;
-  **Note:** As an alternative to these actions, you can select one or multiple topics, then drag and drop them to the desired position inside the map.
- **Root map** - specifies a master DITA Map that Oxygen XML Editor plugin uses to establish a key space which you can use with any other DITA Map that the master one imports.

To watch our video demonstrations about 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.

Editing Actions

-  **Important:** References can be made either by using the href attribute or by using the new keyref attribute to point to a key defined in the map. Oxygen XML Editor plugin tries to resolve both cases. keyrefs are solved relative to the current map.

In addition to the edit actions described above, the contextual menu contains the following actions:

- **Open** - opens in the editor the resources referred by the nodes you select;
- **Append Child/Insert After** - sub-menus containing the following actions:
 - **Reference** - appends/Inserts a topic reference as a child/sibling of the selected node;
 - **New topic** - create a new topic from templates, saves it on disk and adds it into the DITA map;

 **Note:** This action is available for the **Append Child** action.
- **Anchor Reference, Key Definition, Map Reference, Topic Reference, Topic Set, Topic Set Reference** - allows you to insert a reference to a topic file, a map file, a topic set, or a key definition;
- **Topic heading** - appends/Inserts a topic heading as a child/sibling of the selected node;
- **Topic group** - appends/Inserts a topic group as a child/sibling of the selected node;
- **Check Spelling in Files** - checks the spelling of the files in the scope of the current edited map;
- **Search References** - finds in the current map all references to the selected topic reference (topicref or mapref element) and to each element with an ID attribute contained in the selected topic. If the current selection in the **DITA Maps Manager** view is a keydef element the action will find any element with an attribute idref, keyref, conref or conkeyref that points to the selected keydef.
- **Cut, Copy, Paste, Undo, Redo** - common edit actions with the same functionality as those found in the text editor;
- **Paste Before, Paste After** - pastes the content of the clipboard before, respectively after, the selected node;

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 allow you to:

- **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 which ensures both **Promote (Alt + Left Arrow)** and **Demote (Alt + Right Arrow)** operations.

Apart from the above actions, the contextual menu of the root of the map contains the following two options:

- **Open Map in Editor** - opens the ditamap in the **Editor** view;



Note: The bubble displayed on the **Editor** tab of a ditamap opened in the **Editor** view is rendered blue.

- **Open Map in Editor with resolved topics** - opens the ditamap in the editor view and displays the content referred by the links in the ditamap;
- **Export DITA Maps** - exports all the resources referred in a DITA Map. You can also choose to export the resources as an archive.



Tip:



You can open and edit linked topics easily by double clicking the references or by right-clicking and choosing **Open in editor**. If the referenced file does not exist you are allowed to create it.

By right clicking the map root element you can open and edit it in the main editor area for more complex operations.



You can decide to open the reference directly in the Author mode and keep this setting as a default.

Creating a Map

Here are the steps to create a DITA map are the following:

1. Go to menu **File > New** or click on the  **New** toolbar button.
2. Select one of the **DITA Map** templates on the tab **From templates** of the **New** dialog.
3. Click the **OK** button.
A new tab is added in the **DITA Maps Manager** view.
4. Press the  **Save** button on the toolbar of the **DITA Maps Manager** view.
5. Select a location and a file name for the map in the **Save As** dialog.

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.

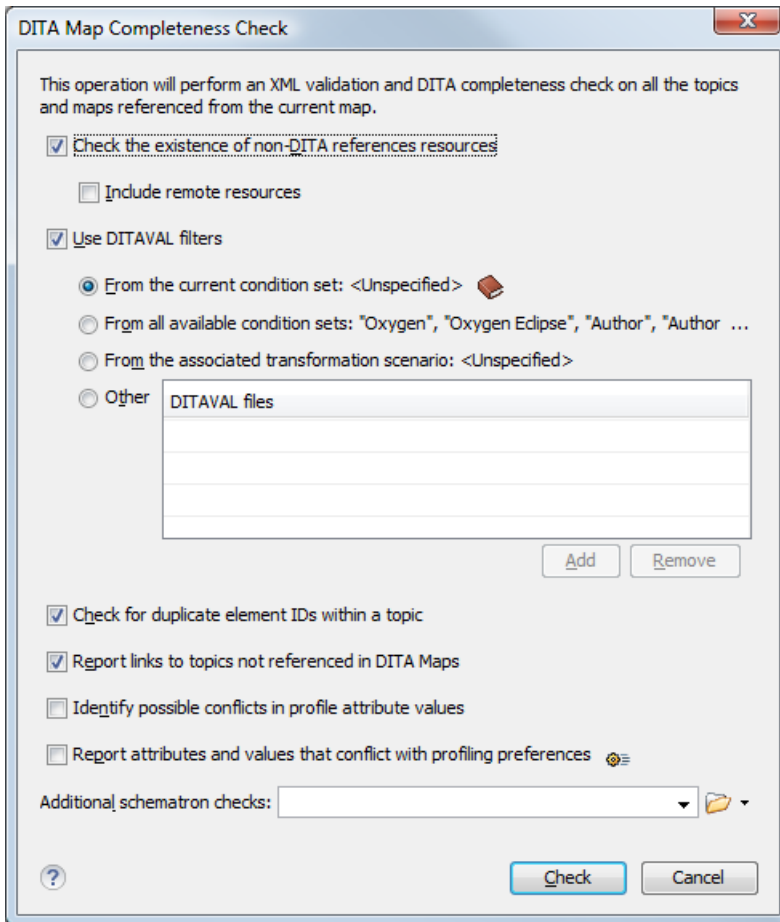


Figure 197: DITA Map Completeness Check

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 referred topic and map. Each topic file is opened and validated against the appropriate DITA DTD. In case another DITA map is referred in the main one, the referred 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:

- **Check the existence of non-DITA references resources** - extends the validation of referred 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 element IDs within a topic** - if an ID is duplicated after assembling all topics referred in the map, it is reported as error;
- **Report links to topics not referenced in DITA Maps** - checks that all referred 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 plugin uses for the validation of DITA resources.

Using a Root Map

You can set a DITA Map as a key space for all the other DITA Maps and topics that you edit in Oxygen XML Editor plugin. This DITA Map is called a *root map*. Specifying the correct root map assures that you encounter no validation problems when you work with keyrefs. All the keys that are defined in a root map are available in the maps that the root map imports.

You can specify the root map from:

- *the DITA Maps Manager view*;
- *the Insert Key Reference Dialog dialog*;
- *the Insert Content Key Reference dialog*.

You can also click one of the key reference errors 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, follow these steps:

1. Run the action **Insert Topic Reference** in the view **DITA Maps Manager**.



The action **Insert Topic Reference** is available on the toolbar and on the contextual menu of the view. The action is available both on the submenu **Append Child** (when you want to insert a topic reference in a map as a child of the current topic reference) and on the submenu **Insert After** (when you want to insert it as a sibling of the current topic reference). The toolbar action is the same as the action from the submenu **Insert After**.

2. Select a topic file in the file system dialog called **Insert Topic Reference**.
3. Press the **Insert** button or the **Insert and close** button in the dialog.
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 in the same file system dialog.
5. Close the dialog using the **Close** button.

Organize Topics in a Map



You can understand better how to organize topics in a DITA map by working with a populated map. You should open the sample map called `flowers.ditamap`, located in the `samples/dita` folder.

1. Open the file `flowers.ditamap`.

2. 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*.
3. 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.
4. Close the map without saving.



Create a Bookmap

The procedure for creating a Bookmap is similar with that for creating a DITA Map.

1. Go to menu **File > New** or click on the  **New** toolbar button.
This action will open *the New wizard*.
2. Select the **DITA Map - Bookmap** template.
3. Click the **OK** button.
A new tab with the new bookmap is added in *the DITA Maps Manager view*.
4. Press the  **Save** button on the toolbar of the **DITA Maps Manager** view.
5. In the **Save As** dialog select a location and a file name for the map.



Create a Subject Scheme

The procedure for creating a DITA subject scheme is similar with that for creating a map.


1. Go to menu **File > New** or click on the  **New** toolbar button.
This action will open *the New wizard*.
2. Select the **DITA Map - Subject Scheme** template.
3. Click the **OK** button.
A new tab with the new subject scheme document is added in *the DITA Maps Manager view*.
4. Press the  **Save** button on the toolbar of the **DITA Maps Manager** view.
5. In the **Save As** dialog select a location and a file name for the map.

Create Relationships Between Topics


The DITA map offers the possibility of grouping different types of links between topics in a relationship table instead of specifying the links of each topic in that topic.

1. Open the DITA map file where you want to create the relationship table.
Use the action  **Open** that is available on the toolbar of the **DITA Maps Manager** view.
2. Place the cursor at the location of the relationship table.
3. Run the action  **Insert a DITA reltable**.
The action is available on the **Author** toolbar, on the menu **DITA > Table** and on the **Table** submenu of the contextual menu of the DITA map editor.
This action displays the **Insert Relationship Table** dialog.
4. Set the parameters of the relationship table that will be created: the number of rows, the number of columns, a table title (optional), a table header (optional).
5. Press **OK** in the **Insert Table** dialog.
6. Set 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.

7. Place the cursor in a table cell and run the action  **Insert Topic Reference** for inserting a topic reference in that cell.

The action is available on the **Author** toolbar, on the menu **DITA > Insert** and on the **Insert** submenu of the contextual menu.

8. Optionally for adding a new row to the table / removing an existing row you should run the action  **Insert Row/ Delete Row**.

The actions are available on the **Author** toolbar, on the menu **DITA > Table** and on the **Table** submenu of the contextual menu.

9. Optionally for adding a new column to the table / removing an existing column you should run the action  **Insert Column/ Delete Column**.

The actions are available on the **Author** toolbar, on the menu **DITA > Table** and on the **Table** submenu of the contextual menu.

Advanced Operations

This section explains how to insert references like chapter, topic reference, topic group or topic heading in a DITA map.

Inserting a Reference, a Key Definition, a Topic Set

A DITA map can contain various types of references. The targets of the references can be:

- an anchor
- a map
- a topic
- a topic set

The `topicref` element is a reference to a topic (such as a concept, task, or reference) or other resource. A `topicref` can contain other `topicref` elements, and allows you to express navigation or table-of-contents hierarchies, as well as implying relationships between the containing `topicref` and its children. You can set the collection type of a container `topicref` to determine how its children are related to each other. You can also express relationships among `topicref`'s using group and table structures (using `topicgroup` and `reltable`). Relationships end up expressed as links in the output (with each participant in a relationship having links to the other participants by default).

A reference to a topic file, a map file, a topic set, or a key definition may be inserted with the following dialog box which is opened from the actions of the **Append child** and **Insert after** submenus of *the DITA Maps Manager view*'s contextual menu. The content of the **Append child** and **Insert after** submenus depend on the selected node of the DITA map tree on which the contextual menu was invoked. For example if the selected node is the bookmark root node the possible child nodes are:

- chapter (the `chapter` element),
- part (the `part` element),
- appendix (the `appendix` element),
- appendices (the `appendices` element)

If the selected node is a `topicref` the possible child nodes are:

- anchor reference (the `anchorref` element),
- topic reference (the `topicref` element),

- map reference (the `mapref` element),
- topic set reference (the `topicsetref` element),
- topic set (the `topicset` element),
- key definition (the `keydef` element),
- topic head (the `topichead` element),
- topic group (the `topicgroup` element)

The same dialog box can be used to insert a non-DITA file like a PDF document.

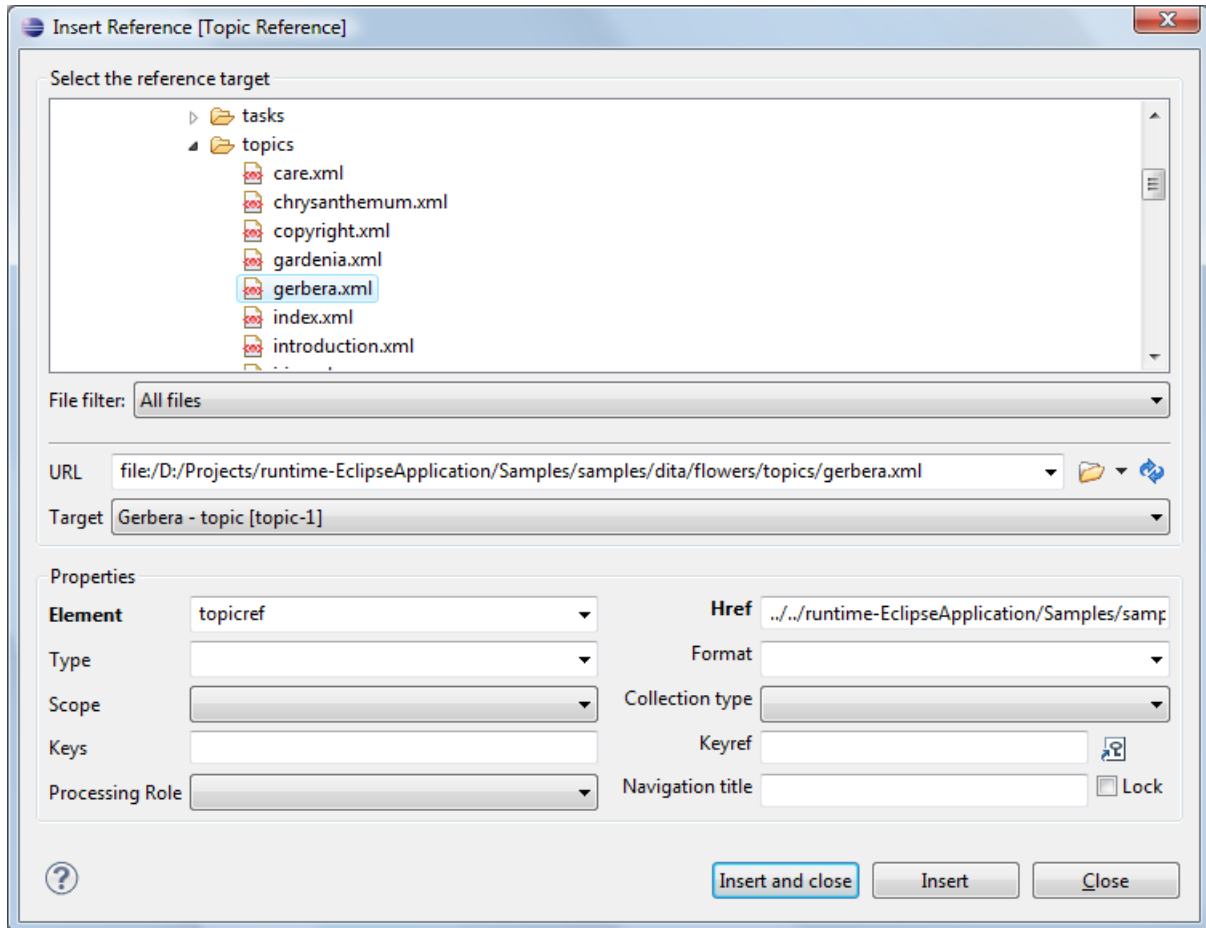



Figure 198: Insert Topic Reference Dialog

By using the **Insert Topic Reference** dialog you can easily browse for and select the source topic file. The **Target** combo box shows all available topics that can be targeted in the file. Selecting a target modifies the **Href** value to point to it which corresponds to the `href` attribute of the inserted `topicref` element. The **Format** and **Scope** combos are automatically filled based on the selected file and correspond to the `format` and `scope` attributes of the inserted `topicref` element. You can specify and enforce a custom navigation title by checking the **Navigation title** checkbox and entering the desired title.

The file chooser located in the dialog allows you to easily select the desired topic. The selected topic file will be added as a child or sibling of the current selected topic reference, depending on the insert action selected from the contextual menu of the **DITA Maps** view, that is an insert action from the **Append child** submenu or from the **Insert after** one. You can easily insert multiple topic references by keeping the dialog 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.

Another easy way to insert a topic reference is to drag files from the **Project** view, file system explorer or **Data Source Explorer** view and drop them into the map tree.

You can also define keys using the **Keys** text field on the inserted `topicref` or `keydef` element. Instead of using the **Href** combo box to point to a location you can reference a key definition using the **Keyref** text field. Use the  **Choose key reference** to access the list of keys defined in the currently open DITA map.

The **Processing Role** combo box allows setting the `processing-role` attribute to one of the allowed values for DITA reference elements: `resource-only`, `normal`, `-dita-use-conref-target`.

Inserting a Topic Heading

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.

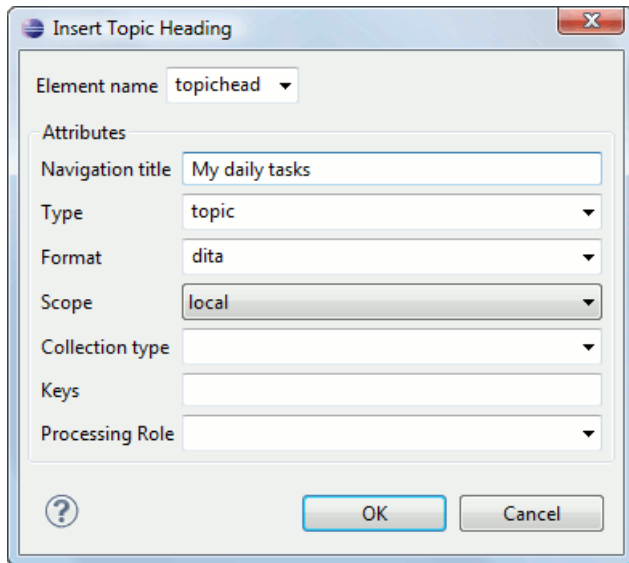


Figure 199: Insert Topic Heading Dialog

By using the **Insert Topic Heading** dialog you can easily insert a `topichead` element. The **Navigation title** is required but other attributes can be specified as well from the dialog.

Inserting a Topic Group

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.

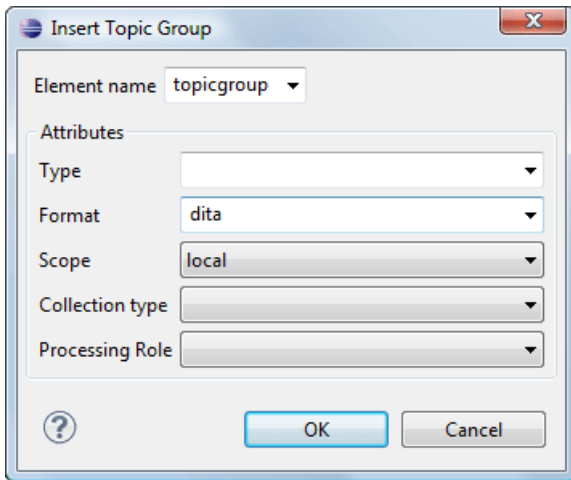


Figure 200: Insert Topic Group Dialog

By using the **Insert Topic Group** dialog you can easily insert a *topicgroup* element. The **Type**, **Format**, **Scope** and **Collection type** attributes can be specified from the dialog.

Edit Properties

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 will show a dialog similar with the *Insert Topic Reference dialog* allowing the editing of some important attributes.
- If a *topichead* element is selected, the action will show a dialog similar with the *Insert Topic Heading dialog* allowing the editing of some important attributes.
- If a *topicgroup* element is selected, the action will show a dialog similar with the *Insert Topic Group dialog* allowing the editing of some important attributes.
- If the map's root element is selected then the user will be able to easily edit the map's title using the **Edit Map title** dialog. By using this dialog you can also specify whether the title will be specified as the *title* attribute to the map or 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 plugin 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 1.6.1 and ANT 1.7 come bundled in Oxygen XML Editor plugin.



More information about the DITA Open Toolkit are available at <http://dita-ot.sourceforge.net/>.

Creating a DITA Transformation Scenario

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**;
- Click the **Configure Transformation Scenario(s)**(**Ctrl (Meta on Mac OS) + Shift + T**) button on the **Transformation** toolbar, then click the **New** button and select **DITA OT Transformation**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **DITA OT Transformation**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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. This dialog presents the list of possible outputs that the **DITA OT Transformation** is able to produce. Select the transformation type, click **OK** and move on to configuring the options in the **New Scenario** dialog. This dialog allows you to configure the options that control the transformation.

All three methods open the **DITA transformation type** dialog box. This dialog presents the list of possible outputs that the **DITA OT Transformation** is able to produce:

- PDF;
- WebHelp;
- WebHelp - Mobile;
- WebHelp with Feedback;
- XHTML;
- Electronic Publication (EPUB);
- *Compiled HTML Help (CHM)*;
- JavaHelp;
- Eclipse Help;
- Eclipse Content;
- *Kindle (DITA 4 Publishers)*;
- HTML2 (DITA 4 Publishers);
- InDesign (DITA 4 Publishers);
- Graphical Map Visualizer (DITA 4 Publishers) - experimental;
- *TocJS*;
- Open Document Format;
- DocBook;
- RTF;
- troff;
- Legacy PDF.

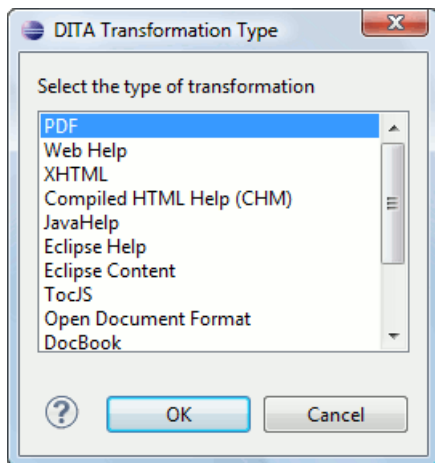


Figure 201: Select DITA Transformation type

Select the transformation type, click **OK** and move on to configuring the options in the **New Scenario** dialog. This dialog allows you to configure the options that control the transformation.



Note: Depending on the type of output you choose, Oxygen XML Editor plugin generates values for the default ANT parameters, so you can execute the scenario right away without further customization.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box allows you to *further customize the DITA OT Transformation scenario* and contains the following tabs:



- *Parameters*;
- *Filters*;
- *Advanced*;
- *Output*;
- *FO Processor*.

Compiled HTML Help (CHM) Output Format

To perform a **Compiled HTML Help (CHM)** transformation Oxygen XML Editor plugin needs `HTML Help Workshop` to be installed on your computer. Oxygen XML Editor plugin automatically detects `HTML Help Workshop` and uses it. When the transformation fails, the `.hlp` (HTML Help Project) file is already generated and it must be compiled to obtain the `.chm` file. Note that `HTML Help Workshop` fails when the files used for transformation contain diacritics in their names, due to different encodings used when writing the `.hlp` and `.hnc` files.

Changing the Output Encoding

Oxygen XML Editor plugin uses the `html.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 plugin has the `html.locale` parameter set to `en-US`. The format of the `html.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>.

The default value of the `html.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 `html.locale` parameter and change its value to the desired language tag.

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 webpage 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.

The TocJS Transformation

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 `${frameworks}/dita/DITA-OT/plugins/com.sophos.tocjs/basefiles` folder in the transformation's output folder.

3. Copy the `${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"`.

Kindle Output Format

Enabling Oxygen XML Editor plugin to obtain Kindle output from DITA Maps, requires KindleGen. To download KindleGen, go to www.amazon.com/kindleformat/kindlegen and select the zip file that matches your operating system. Unzip the file on your local disk. After you download and unzip KindleGen, open the **Kindle** transformation type, select **Edit**, and go to the **Parameters** tab. Set the **kindlegen.executable** parameter as the path to the KindleGen directory.


Migrating OOXML Documents to DITA

Oxygen XML Editor plugin 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 plugin. 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. In case you encounter any issues with the transformation, click the link below for further details about the Word to DITA Transformation Framework.

Customizing a DITA Scenario

This section explains how to customize the parameters of a DITA transformation scenario.

The Parameters Tab

This dialog allows you to configure the parameters sent to the DITA-OT build file.

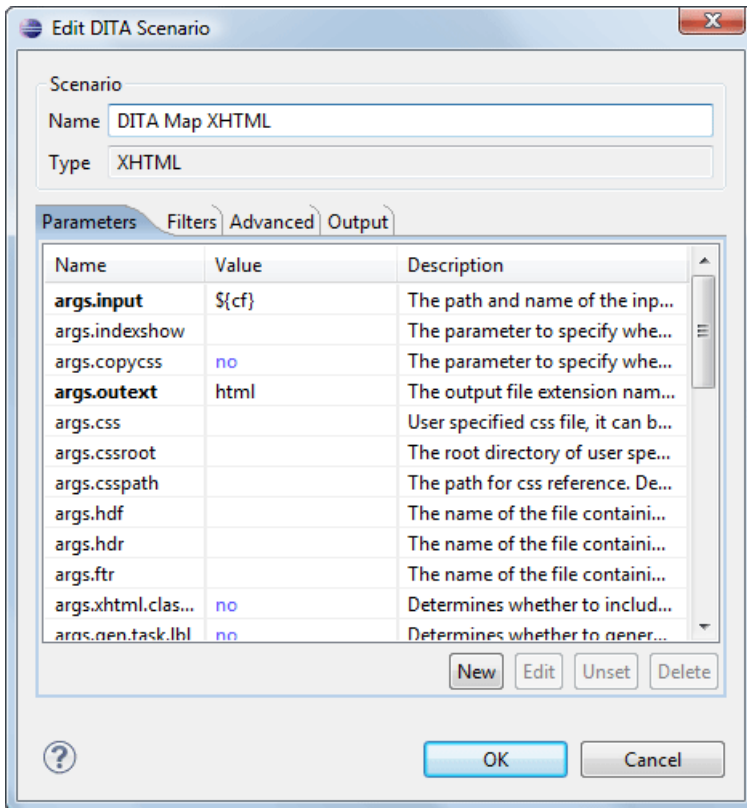


Figure 202: Edit DITA Ant transformation parameters

All the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (eg: XHTML) are listed along with their description. The values for some important parameters are already filled in. You can find more information about each parameter in the [DITA OT Documentation](#). You can also add additional parameters to the list.

Using the toolbar buttons you can add, edit or remove a parameter.

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 variables selector to simplify setting a file path as value to a parameter.

The value of a parameter can be entered at runtime if a value `ask('user-message', param-type, 'default-value' ?)` is used as value of parameter in the Configure parameters dialog.

Examples:

- `${ask('message')}` - Only the message displayed for the user is specified.
- `${ask('message', generic, 'default')}` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
- `${ask('message', password)}` - 'message' is displayed, the characters typed are masked with a circle symbol.
- `${ask('message', password, 'default')}` - same as before, the default value is 'default'.
- `${ask('message', url)}` - 'message' is displayed, the parameter type is URL.
- `${ask('message', url, 'default')}` - same as before, the default value is 'default'.

The Filters Tab

In the scenario **Filters** tab you can add filters to remove certain content elements from the generated output.

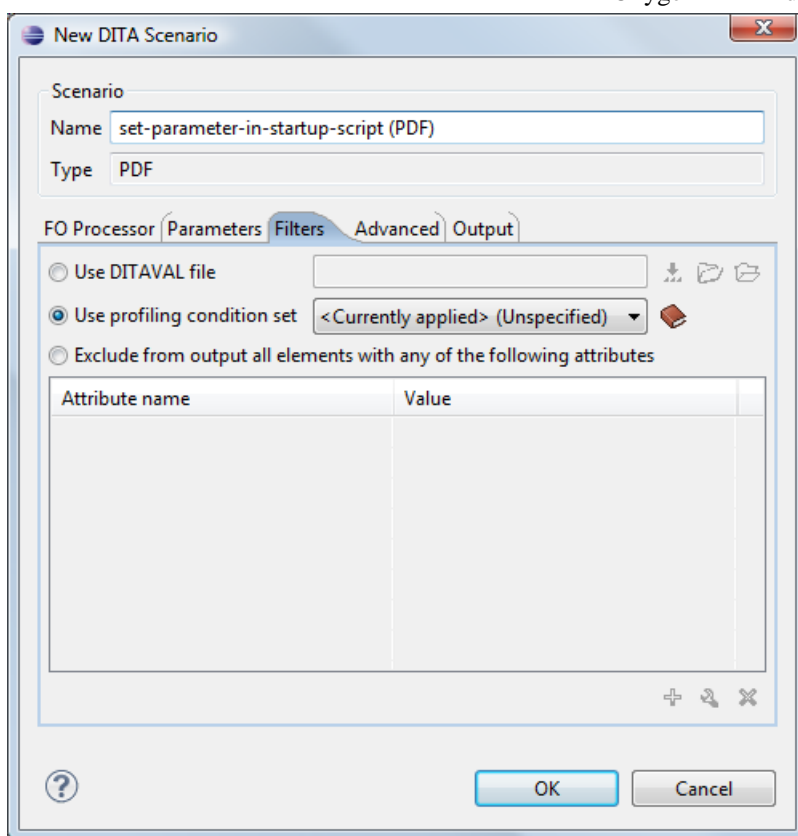


Figure 203: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - if you already have a DITAVAL file associated with the transformed map you can specify the path to it and it will be used when filtering content. You can find out more about constructing a DITAVAL file in the [DITA OT Documentation](#) topic;
- **Use profiling condition set** - sets the *profiling condition set* that applies to the document you transform;
- **Exclude from output all elements with any of the following attributes** - you can configure a simple list of attribute (name, value) pairs which when present on an element in the input will remove it from output.

The *Advanced* Tab

In the **Advanced** tab, you can specify advanced options for the transformation.

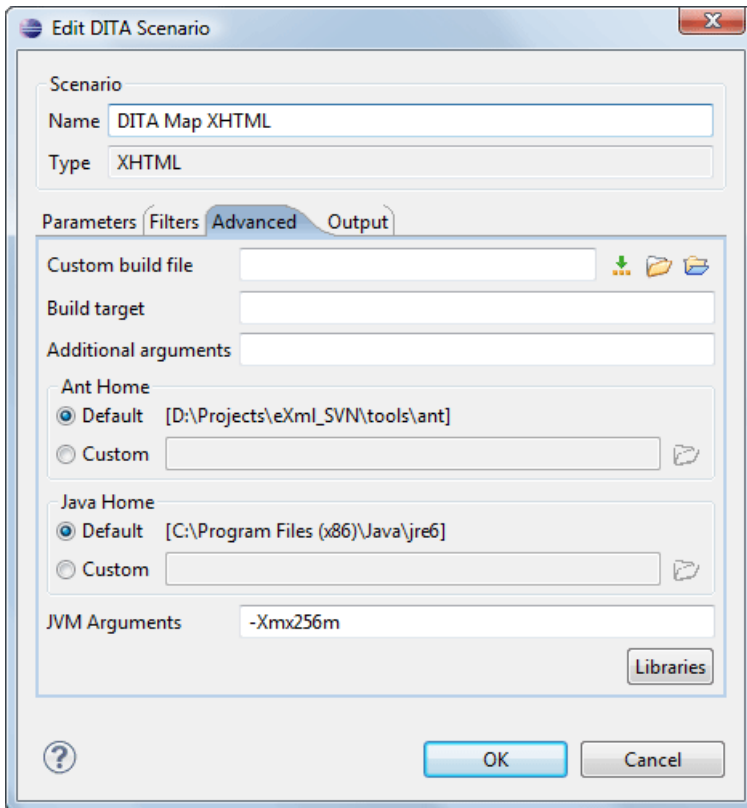


Figure 204: Advanced settings tab

You have several parameters that you can specify here:

- **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` directory configured in the **Parameters** tab is used.
- **Build target** - You can specify a build target to the build file. By default no target is necessary and the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation like `-verbose`.
- **Ant Home** - You can specify a custom ANT installation to run the DITA Map transformation. By default it is the ANT installation bundled with Oxygen XML Editor plugin.
- **Java Home** - You can specify a custom Java Virtual Machine to run the ANT transformation. By default it is the Java Virtual Machine used by Oxygen XML Editor plugin.
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to `-Xmx384m` which means the transformation process is allowed to use 384 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (384 MB) to a higher value, like 512 MB. This way, you can avoid the Out of Memory error messages (**OutOfMemoryError**) received from the ANT process.



Note: If you are publishing DITA to PDF and still experience problems, you should also increase the amount of memory allocated to the FO transformer. To do this, go to the **Advanced** tab and increase the value of the **Java Arguments** parameter.

- **Libraries** - Oxygen XML Editor plugin adds by default as high priority libraries which are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can specify all the additional libraries (jar files or additional class paths) which are used by the ANT transformer. You can also decide to control all libraries added to the classpath.

The Output Tab

In the **Output** tab, you can configure options related to the place where the output is generated.

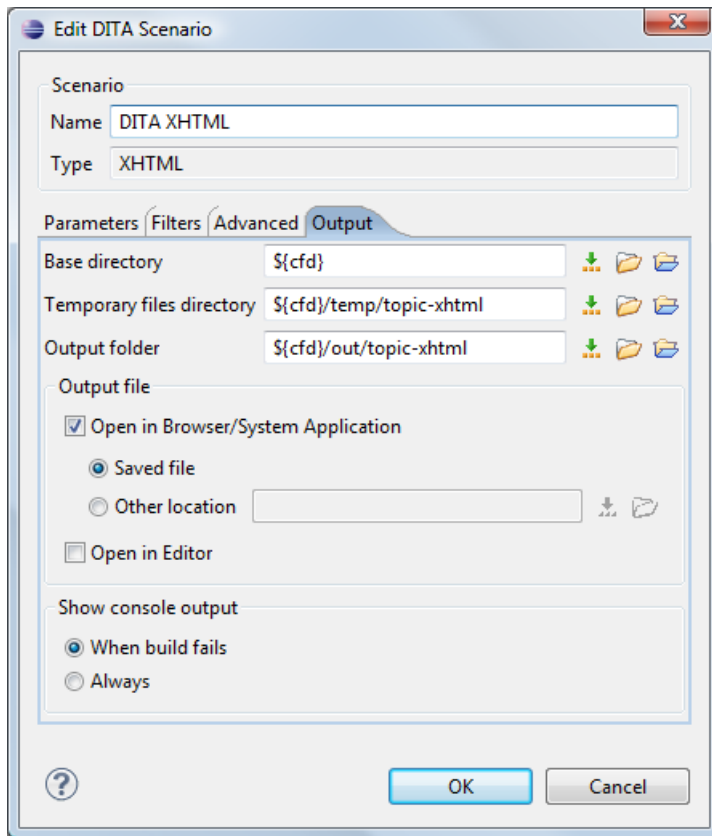


Figure 205: Output settings tab

You have several parameters that you can specify here:

- **Base directory** - all the relative paths which appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located;
- **Temporary files directory** - this directory is used to store pre-processed temporary files until the final output is obtained;
- **Output folder** - the folder where the final output content is copied;
- **Output file options** - the transformation output can then be opened in a browser or even in the editor, if specified;
- **Show console output** - specifies whether the console is always displayed or only when the build fails.



Note: If the DITA Map or topic is opened from a remote location or from a ZIP file, the scenario must specify absolute output, temporary and base file paths.

The FO Processor Tab

This tab allows you to set an FO Processor, when you choose to generate PDF output.

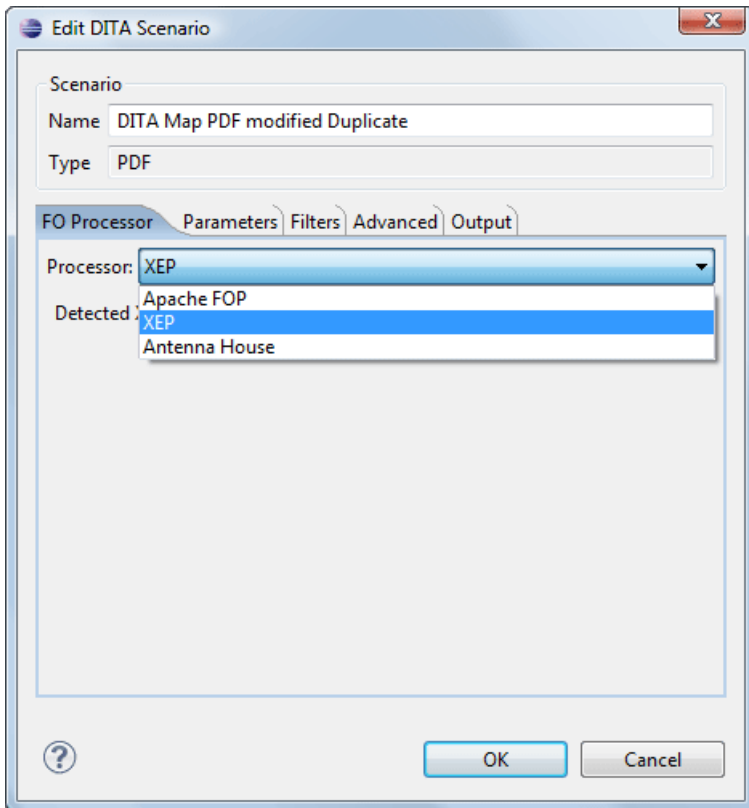


Figure 206: FO Processor configuration tab

You can choose between:

- **Apache FOP** - Default setting. This processor comes bundled with Oxygen XML Editor plugin.
- **XEP** - The *RenderX* XEP processor.

If you select **XEP** in the combo and XEP was already installed in Oxygen XML Editor plugin you can see the detected installation path appear under the combo box.

XEP is considered as 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 (eg: `xep.bat` on Windows);
- XEP was installed in the `frameworks/dita/DITA-OT/plugins/org.dita.pdf2/lib` directory of the Oxygen XML Editor plugin installation directory.
- **Antenna House** - The *Antenna House* AH (v5) or XSL (v4) Formatter processor.


If Antenna House was already installed on your computer and you select **Antenna House** in the combo box, in Oxygen XML Editor plugin you can see the detected installation path appear under the combo.

Antenna House is considered as 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 plugin preferences pages.

Running a DITA Map ANT Transformation

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.

 **Tip:** The HTTP proxy settings are also used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the Network Connections.

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 then a font that is capable to render 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](#).

How does the DITA Open Toolkit PDF font mapping work?

The DITA OT contains a file

`OXYGEN_INSTALL_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 XSLs, the processing defaults to **Helvetica**.

Tips and Tricks

This section contains solutions for common problems encountered when working with the DITA Open Toolkit.

Debugging PDF Transformations

To debug a DITA PDF transformation scenario using the XSLT Debugger follow these steps:

1. Go to **Options > Preferences > XML > XML Catalog**, click **Add** and select the file located at [Oxygen Install Directory]\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 in Oxygen XML 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.xsl` stylesheet located at
`OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl`;
7. 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. It's value should have an URL syntax
`like:file:///c:/path/to/OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg.`
8. Debug the transformation scenario.

The PDF Processing Fails to Use the DITA OT and Apache FOP

There are cases when publishing DITA content which you consider perfectly valid fails when creating the PDF file. This topic tries to list some of the more usual situations and has fix-up hints for each of these cases.

- 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] at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:647)
[fop] at org.apache.fop.tools.anttasks.FOPTaskStarter.render(Fop.java:676)
.....
[fop] at org.apache.tools.ant.launch.Launcher.run(Launcher.java:280)
[fop] at org.apache.tools.ant.launch.Launcher.main(Launcher.java:109)
[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)
[fop] at java.io.FileOutputStream.open(Native Method)
[fop] at java.io.FileOutputStream.<init>(Unknown Source)
[fop] at java.io.FileOutputStream.<init>(Unknown Source)
[fop] at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:644)
[fop] ... 45 more
[fop] C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo -> C:\samples\dita\out\pdf\test.pdf
```



Tip: Such an error message usually means that the PDF file is already opened in a PDF Reader application. The workaround is simple, always close the external PDF reader application 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\exml\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.anttasks.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
```



Tip:

- Find the DITA topic that contains a DITA CALS table for which one of the table rows has more cells than the number of `colspecs` defined on the table.
 - Edit the transformation scenario you and set the parameter `clean.temp` to `no`.
 - Run the transformation, open the `topic.fo` file in Oxygen XML Editor plugin, and look in it at the line specified in the error message (See `position 179:-1`).
 - 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.
```

**Tip:**

- Use the **Validate and Check for Completeness** action available in the **DITA Maps Manager** view to find such problems.
- In case you publish to PDF using a certain DITAVAL filter, set the same DITAVAL file in the **DITA Map Completeness Check** dialog.
- In case the **Validate and Check for Completeness** action does not discover any issues, edit the transformation scenario and set the *clean.temp* parameter to *no*.
- Run the transformation, open the `topic.fo` file in Oxygen XML Editor plugin, and search in it for the `unique_4_Connect_42_wrongID` id.
- 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.

Topic References outside the main DITA Map folder

Referencing to a DITA topic, map or to a binary resource (eg: 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.

Embedding videos in the WebHelp output

The DITA *object* element maps precisely to an HTML `object` element. So you can try directly to construct a simple HTML page with an `object` element pointing to the video and after you get it working in the browser you can use the same code in the DITA content.

For example, in order to embed a YouTube video in the HTML output the DITA content should contain the following construct:

```
<object width="425" height="349" type="application/x-shockwave-flash" data="http://www.youtube.com/v/hXsgbBwbr7M">
  <param name="movie" value="http://www.youtube.com/watch/v/hXsgbBwbr7M" />
</object>
```

Syntax Highlight Inside Codeblock Sections

Codeblock elements can be used in the DITA content to give examples from different programming or scripting languages like XML, CSS, Java and so on.

You can add to the XHTML-based and PDF published outputs syntax highlight inside these codeblock sections, by setting the *outputclass* attribute on the *codeblock* element to a specific value depending on the content type:

- `language-xml;`
- `language-java;`
- `language-css;`
- `language-javascript;`
- `language-sql;`
- `language-c;`
- `language-cpp;`

- language-csharp;
- language-ini;
- language-php;
- language-python;
- language-ruby;
- language-perl;
- language-bourne.

To customize different colors for the highlighted tokens you can edit the XSLT stylesheets:

- OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight/pdfHighlight.xsl
- OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight/xhtmlHighlight.xsl

DITA-OT Customization

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 plugin Ant Tool

The Ant 1.7 tool which comes with Oxygen XML Editor plugin is located in the [Oxygen-install-folder]/tools/ant directory. Any additional libraries for Ant must be copied to the Oxygen XML Editor plugin Ant lib directory.

If you are using Java 1.6 to run Oxygen XML Editor plugin 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 refer topics using URI's 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 plugin](#). 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_INSTALL_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 plugin 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 plugin and create the template called createFrontMatter_1.0. This will override the same template from the OXYGEN_INSTALL_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: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 plugin.

Header and Footer Customization

The XSLT stylesheet

OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/static-content.xml 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

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 321.

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 from the used DITA Open Toolkit installation (by default `OXYGEN_INSTALL_DIR\frameworks\dita\DITA-OT\plugins`).
2. The DITA OT ANT integrator build file needs to be run. In the **Transformation Scenarios** view there is a predefined transformation scenario called **Run DITA OT Integrator** which can be used for this.



Important: The folder where the DITA OT is located needs to have full write access permissions set to it.

3. If the plugin contributed with a new **transtype** to the publishing stage, the application will not detect it by default. You have to create a new **DITA OT** transformation scenario with a predefined type which is close to the newly

added **transtype**, then edit the transformation scenario and in the **Parameters** tab add a parameter called **transtype** with the value of the newly added transformation type.

Creating a Simple DITA OT HTML and PDF Customization Plugin

This example describes a **DITA Open Toolkit** plugin which you can use to provide syntax highlight when publishing DITA **codeblock** elements to **PDF** or **HTML**-based outputs. The plugin is available in the DITA Open Toolkit distribution which comes with the application.

Here are some steps to help anyone wanting to create an **XSLT** customization plugin for the **DITA Open Toolkit** for HTML and PDF based outputs.

1. Create a folder for your plugin in the DITA OT **plugins** folder. The **DITA OT** bundled with Oxygen can be found here:

```
OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT
```

In my case I created the following folder:

```
OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight
```

2. Create a **plugin.xml** file in that folder containing the extension points of the plugin. In my case, the plugin descriptor file contains:

```
<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.xml.xhtml" value="xhtmlHighlight.xml" type="file"/>
  <feature extension="dita.xml.xslfo" value="pdfHighlight.xml" type="file"/>
</plugin>
```

The important extensions in it are the references to the XSLT stylesheets which will be used to style the HTML and the PDF outputs.

You can find a bunch of 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 called **xhtmlHighlight.xml** located in the same plugin folder.

As I want to overwrite the creation of the HTML content from a DITA **codeblock** element I will first need to find the XSLT template that I need to overwrite. A DITA **codeblock** element has the **class** attribute value **" + topic/pre pr-d/codeblock "**. Usually in such cases I take part of the class attribute value and search using the **"Find/Replace in Files"** Oxygen action in all of the DITA OT XSLT resources.

In this case I searched for **topic/pre** and found this XSLT stylesheet:

```
OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/xsl/xslhtml/dita2htmlImpl.xml
```

containing this XSLT template:

```
<xsl:template match="*[contains(@class,' topic/pre ')]" name="topic.pre">
  <xsl:apply-templates select="." mode="pre-fmt" />
</xsl:template>
```

thus my **xhtmlHighlight.xml** will overwrite the content of the template like:

```
<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,' ditaot-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>
```

and call another XSLT template which applies as a Java extension the XSLTHL library to style the content.

4. Create an XSLT stylesheet called **pdfHighlight.xml** located in the same plugin folder which will contain the PDF XSLT customization. In this case I will overwrite the XSLT template from:

OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/legacypdf/xslfo/dita2fo-elems.xml

which has the content:

```
<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>
```

5. To install your plugin in the DITA OT, run the integrator. In the application's **Transformation Scenarios** view, enable the **Show all scenarios** action, available in the drop down settings button. Just check that and execute the transformation scenario called **Run DITA OT Integrator**.

And that's it, your XSLT content will be applied with priority when publishing both to XHTML-based (WebHelp, CHM, EPUB, JavaHelp, Eclipse Help) and to PDF-based outputs.

Now, let's take a look at what the step (5) - running the integrator to install the plugin - really did:

1. In the XSLT stylesheet:

OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/xsl/dita2html-base.xml

a new import automatically appeared:

```
<xsl:import href="../../plugins/com.oxygenxml.highlight/xhtmlHighlight.xml"/>
```

This import is placed after all base imports and because of this it has a higher priority. More about imported template precedence can be found in the XSLT specs:

<http://www.w3.org/TR/xslt#import>

2. Likewise, in the top-level stylesheets related to PDF publishing like:

OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xml

a new import statement has appeared:

```
<xsl:import href="../../../../com.oxygenxml.highlight/pdfHighlight.xml"/>
```

Now, you can distribute your plugin folder to anyone having 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.

DITA Specialization Support

This section explains how you can integrate and edit a DITA specialization in Oxygen XML Editor plugin.

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, you should copy the plugin to the place where the DITA OT that you are using is located (by default `OXYGEN_INSTALL_DIR\frameworks\dita\DITA-OT\plugins`). Then you should 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 folder 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:

If the DTD's that define the extension elements are located in a folder outside the DITA Open Toolkit folder you should 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 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 page.

The content completion should work without additional modifications and you can choose the tags which 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 if this is the case. 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 a New DITA Open Toolkit in Oxygen XML Editor plugin

Oxygen XML Editor plugin comes bundled with a DITA Open Toolkit, located in the `[Oxygen-install-folder]/frameworks/dita/DITA-OT` directory. To use a new DITA Open Toolkit, follow these steps:

1. Edit your transformation scenarios and in the **Parameters** tab change the value for the `dita.dir` directory to point to the new directory.
2. To make changes in the libraries that come with the new 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 DTD's and you want to use the new versions for content completion and validation, go to the Oxygen XML Editor plugin 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

The DITA framework allows reusing 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_INSTALL_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 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 referred by the element but is located in a different file. You have the option of displaying referenced content by setting the option **Display referred content** that is available from menu **Options > Preferences > Editor > Edit modes > Author**.

Working with Content References

The `conref` DITA feature (short for *content reference*) lets you include a piece of source content as reference in other contexts. 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 plugin 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 plugin 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 `tbody` or a `tbody` 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 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 to save the reusable component in a file. If the checkbox was selected in the **Create Reusable Component** dialog 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. Just select in the file system dialog 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 open 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 refer 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 refer 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 is displayed.
4. In the **Insert Content Reference** dialog, from the **URL** field, navigate to the topic that holds the element you want to refer.
In the **Target ID** section of the **Insert Content Reference** dialog Oxygen XML Editor plugin presents the elements that you can refer.
5. Click the ID of the element you want to refer, 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.



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.

The Insert Content Reference Dialog

The **Insert Content Reference** dialog lets you reuse content by inserting references to the DITA elements that hold the content you want to reuse.






Note: To refer 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 plugin *displays the referred 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 plugin*. If the URI's can be resolved, the referred content is displayed in the **Author** mode and in the transformation output.

To open the **Insert Content Reference** dialog, 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**.

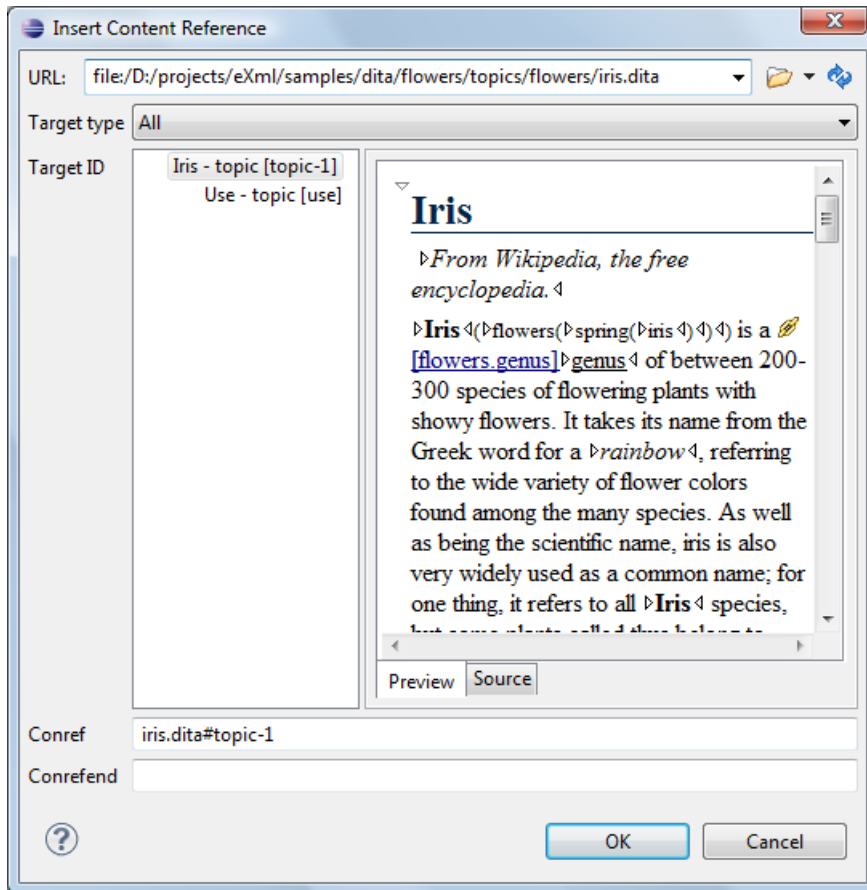



Figure 207: The Insert Content Reference Dialog

 **Note:** The **Insert Content Reference** dialog is not modal. The dialog is closed automatically in case you switch to a different editor.

The following fields are available in this dialog:

- **URL** - specifies the path to the topic that holds the content you want to refer;
- **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 refer;
- **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.

Moving and Renaming Resources

You can move or rename resources on disk directly from Oxygen XML Editor plugin. 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. This dialog 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 referred 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 plugin 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. This dialog 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 referred 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 plugin 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 plugin 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 plugin 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://oxygenxml.com/demo/DITA_Profiling.html.

Profiling / Conditional Text Markers

If **Show Profiling Attributes** option (available in the  *Profiling / Conditional Text toolbar menu*) is set 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.

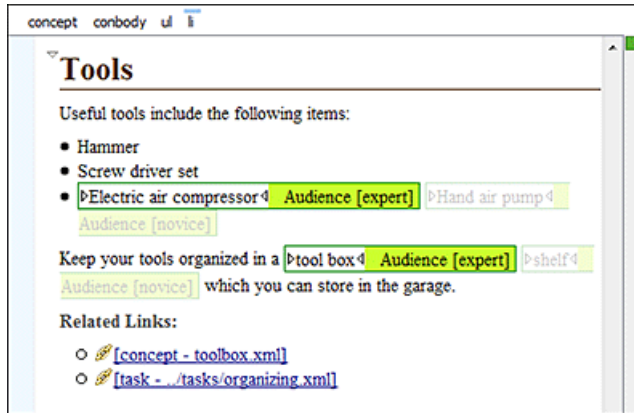





Figure 208: Profiling in Author

In the *DITA Maps Manager View* different decorators 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.

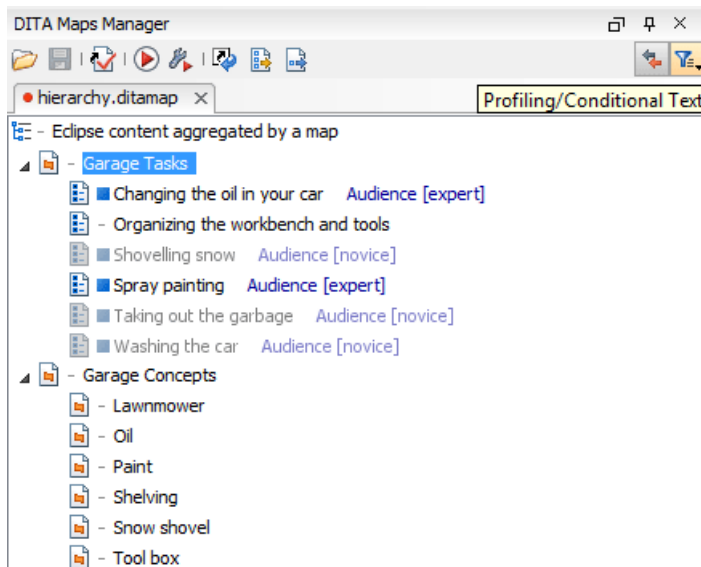


Figure 209: Profiling in DITA Maps Manager

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 instead of a collection of topics. The 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 is 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:

```
<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** mode or in the **Author** mode, Oxygen XML Editor plugin collects all the profiling values from the Subject Scheme Map that is referenced in the map 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 our example the values `therapist`, `oncologist` up to `plastic-surgeon` 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>
```

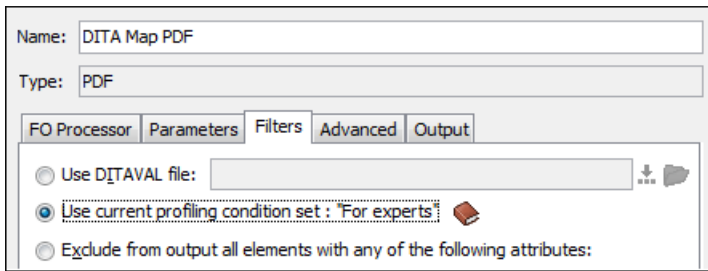
When you define a DITAVAL filter you can exclude for instance anything which is `surgeon`:

```
<val>
  <prop action="exclude" att="audience" val="surgeon"/>
</val>
```

Then if you transform the main DITA Map specifying the DITAVAL filter file in the transformation scenario the `p` element should be excluded from the output. So excluding the `surgeon` audience excludes also the `neuro-surgeon` and `plastic-surgeon` from the output.

Publish Profiled Text

Oxygen XML Editor plugin comes with preconfigured transformation scenarios for DITA. All these scenarios take into account the current profiling condition set.




How to Profile DITA Content

1. Go to **Options > Preferences > Editor > Edit modes > Author > Profiling / Conditional Text** page and edit the **Profiling Attributes** table.
2. For DITA there are already default entries for `audience`, `platform`, `product` and `otherprops`. 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 (Ctrl (Meta on Mac OS)+ click on MacOS) a topic reference in **DITA Maps Manager** and choose **Edit Profiling Attributes** from the contextual menu.
 - b) In the **Author** editing mode, right-click (Ctrl (Meta on Mac OS)+ click on MacOS) an XML element and choose **Edit Profiling Attributes** from the contextual menu.
 - c) Use the **Attributes** view to set profiling attributes.
 Turn on 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 in the **New** file action wizard.
- reference an external MathML file as an image, using the  **Insert Image Reference** toolbar action.

Note that 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 users must employ additional customizations for handling MathML content.

MathML Equations in the HTML Output

MathJax is a solution to properly view **MathML** equations embedded in **HTML** content in a variety of browsers.

Without the help of the **MathJax Javascript** code, right now only **Firefox** can render **MathML** equations embedded in the **HTML** code.

Let's say you have **Docbook** or **DITA** content which has embedded **MathML** equations and you want to properly view the equations in the published HTML output types (WebHelp, CHM, EPUB and so on).

You need to add a reference to the MathJax script in the **head** of all HTML files which have the equation embedded in them:

```
<script type="text/javascript"
src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML"></script>
```

For DITA you can 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 like IE, Chrome and Firefox.

Chapter 7

Predefined Document Types

Topics:

- [Document Type](#)
- [The DocBook 4 Document Type](#)
- [The DocBook 5 Document Type](#)
- [The DocBook Targetset Document Type](#)
- [The DITA Topics Document Type](#)
- [The DITA Map Document Type](#)
- [The XHTML Document Type](#)
- [The TEI ODD Document Type](#)
- [The TEI P4 Document Type](#)
- [The TEI P5 Document Type](#)
- [The EPUB Document Type](#)

The following are the short presentations of some document types that come bundled with Oxygen XML Editor plugin. For each document type there are presented built-in transformation scenarios, document templates and Author extension actions.

Document Type

A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the **Author** mode for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both **Author** mode and Text mode;
- CSS stylesheet(s) for rendering XML documents in **Author** mode;
- user actions invoked from toolbar or menu in **Author** mode;
- predefined scenarios used for transformation of 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.

The tagless editor comes with some predefined document types already configured when the application is installed on the computer. These document types describe well-known XML frameworks largely used today for authoring XML documents. Editing a document which conforms to one of these types is as easy as opening it or creating it from one of the predefined document templates which also come with Oxygen XML Editor plugin.

To see our video demonstration about configuring a framework in Oxygen XML Editor plugin, go to <http://oxygenxml.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:

- root element name is `book` or `article`;
- the PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`.

The schema of *DocBook 4* documents is `#{frameworks}/docbook/dtd/docbookx.dtd`, where `#{frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DocBook content is located in `#{frameworks}/docbook/css/docbook.css`.

The XML catalog is stored in `#{frameworks}/docbook/catalog.xml`.

To watch our video demonstration about editing DocBook documents, go to http://oxygenxml.com/demo/DocBook_Editing_in_Author.html.

DocBook 4 Author Extensions

Specific actions for DocBook documents are:

- **B Bold emphasized text** - emphasizes the selected text by surrounding it with `<emphasis role="bold"/>` tag;
- *I Italic emphasized text* - emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag;
- U Underline emphasized text - emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag.

**Note:**

Bold, **Italic**, and **Underline** are toggle actions.

For all of the above actions, if there is no selection, then a new `emphasis` tag with specific role is inserted. These actions are available in any document context and are grouped under the **Emphasize** toolbar actions group.

- **Browse reference manual** - opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position;
- **Cross reference (link)** - inserts a hypertext link;
- **Cross reference (xref)** - inserts a cross reference to another part of the document;



Note: These actions are grouped under the **Link** toolbar actions group.

- **Web Link (ulink)** - inserts a link that addresses its target with an URL (Universal Resource Locator);
- **Insert olink** - inserts a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values which are present in a *Targetset* file;

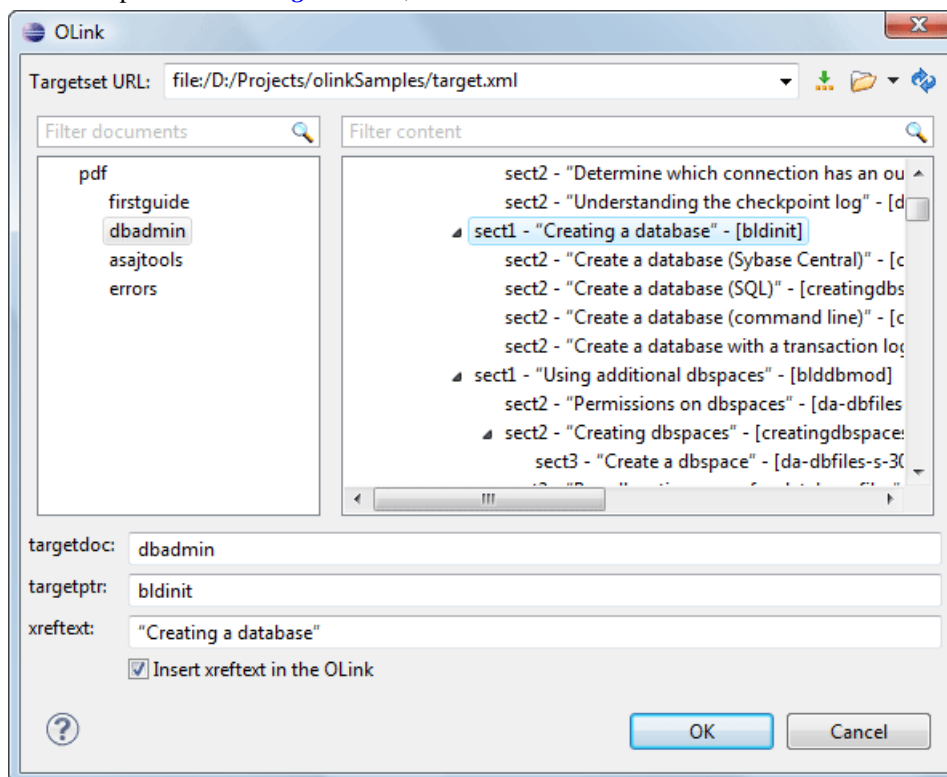



























Figure 210: Insert OLink Dialog

After you choose the **Targetset** URL, the structure of the target documents is presented. For each target document (`targetdoc`), the content is displayed allowing for easy identification of the `targetptr` for the `olink` element which 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 also have the possibility to edit an `olink` using the action **Edit OLink** 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 Section** - inserts a new section / subsection in the document, depending on the current context. For example if the current context is `sect1` then a `sect2` is inserted, and so on;
- **← 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 image reference** - inserts a graphic object at the caret position. This is done by inserting either <figure> or <inlinegraphic> element depending on the current context. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG;
 -  **Insert an ordered list at the caret position** - inserts an ordered list. A child list item is also inserted automatically by default;
 -  **Insert an unordered list at the caret position** - inserts an itemized list. A child list item is also inserted automatically by default;
 -  **Insert a step or list item** - inserts a new list item in any of the above list types.
 -  **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;
 -  **Insert Table** - opens a dialog 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 plugin 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;
 -  **Insert Row Above** - inserts a row above the current one;
 - **Insert Row Below** - inserts a row below the current one;
 -  **Insert Column Before** - inserts a column before the current one;
 - **Insert Column After** - inserts a column after the current one;
 -  **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 plugin deletes the source row is 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 plugin 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 plugin 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 plugin 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 plugin decreases the column span of the source cell with one.



Caution: Column specifications are required for table actions to work properly.

- **Generate IDs** - Available in the contextual menu, this action allows you to generate an unique ID for the element at caret position. If the element already has an ID set, it is preserved.

Further options are offered in the **ID Options** dialog, available in the document type specific main menu. The configurable ID value pattern can accept most of the application supported *editor variables*. You can also specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled.

If you want to keep an already set element ID's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.

All actions described above are available in the contextual menu, the **DocBook4** submenu of the main menu or in the **Author custom actions** toolbar.

Dragging a file from *the Project view* or from *the 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) with the location of the dragged file at the drop location (similar with the **Insert Graphic** toolbar action).

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp, EPUB and EPUB 3.

WebHelp Output Format

WebHelp is a form of online help consisting 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.

Oxygen XML Editor plugin allows you to publish DocBook 4 documents into a WebHelp format which provides both table of contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;




Note: In case your documents contain no *indexterm* elements, the **Index** tab is not generated.



Note: You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

- the right frame 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.

You can use this button , displayed in the **Content** tab, to collapse all the topics presented 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 presents the table of contents in the left side and a section that presents the content of a topic in the right side;

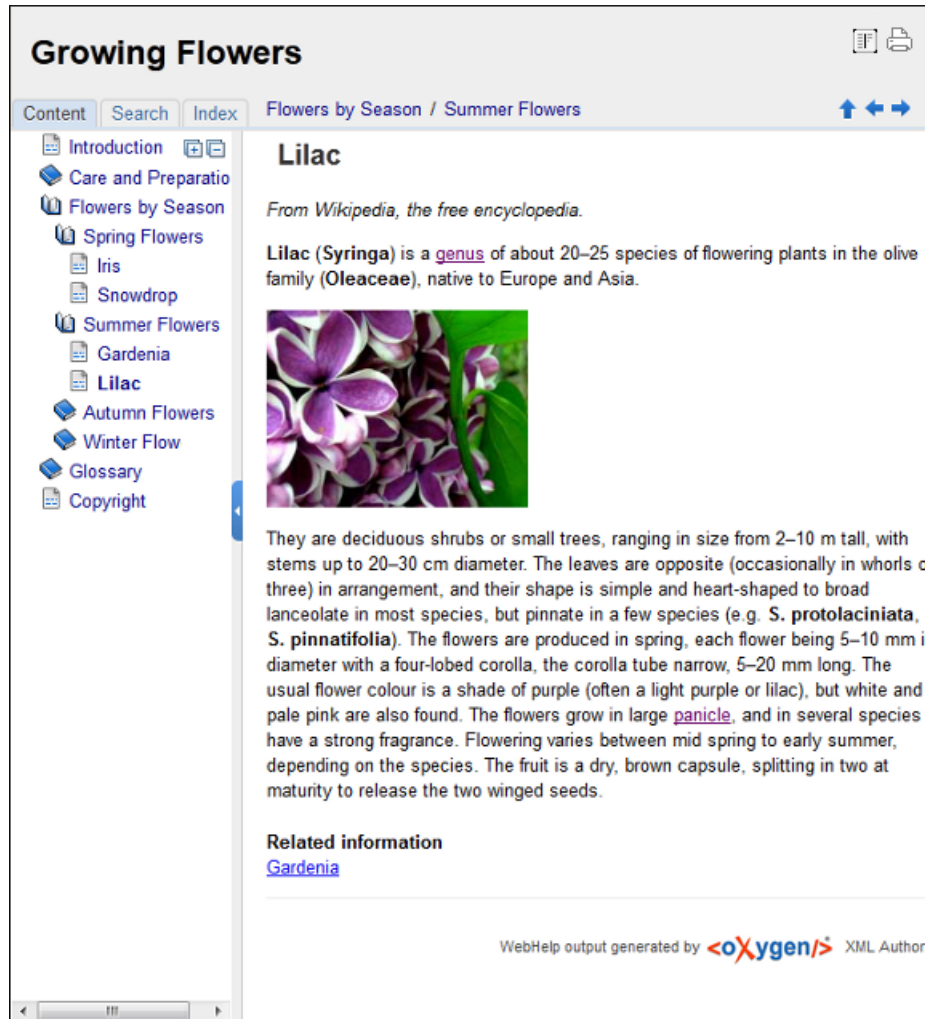


Figure 211: WebHelp Output

To publish DocBook 4 to WebHelp, use the **DocBook WebHelp** transformation. To further customize the out-of-the-box transformation, you can edit some of its parameters:

- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. Default setting is `false`;
- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `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, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on;
- `xml.file` - this parameter specifies the path to the DocBook XML file.

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:

- number of keywords found in a single page. The higher the number, the better;

- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like "*grow flowers*", returns no results in our case, because it searches for two separate words: "*grow* and *flowers*" (note the quote signs attached to each word);
- *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 a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.



Note: This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;
- Opera.

WebHelp with Feedback Output Format

This section presents the Feedback-Enabled WebHelp systems support.

Oxygen XML Editor plugin has the ability to transform DocBook documents into Feedback-Enabled WebHelp systems. WebHelp is a form of online help consisting 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. The Feedback system allows you to view discussion threads in a tree-like representation, reply to already posted comments and use stylized comments.

Oxygen XML Editor plugin allows you to publish DocBook 4 documents into a WebHelp with Feedback format which provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;



Note: In case your documents contain no *indexterm* elements, the **Index** tab is not generated.



Note: You can choose to enhance the appearance of the selected item in the table of contents. The [WebHelp customization](#) topic contains more details about this.

- the right frame 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.

To publish DocBook 4 to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation scenario. To further customize the out-of-the-box transformation, you can edit some of its parameters such as:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. 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, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on;
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server.

For further information about all the DocBook transformation parameters, go to <http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

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:

- number of keywords found in a single page. The higher the number, the better;
- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"grow flowers"*, returns no results in our case, because it searches for two separate words: *"grow* and *flowers"* (note the quote signs attached to each word);
- *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 a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

To watch our video demonstration about the feedback-enabled WebHelp system, go to http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.



Note: 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 you to load WebHelp pages in Google Chrome only from a web server.



Note: In case you need to automate the transformation process and use it outside of Oxygen XML Editor plugin, you can use [the Oxygen WebHelp plugin](#).

Introduction

Oxygen XML Editor plugin has the ability to transform DocBook 4 documents into feedback-enabled WebHelp systems.

WebHelp is a form of online help consisting 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. It also provides table of contents and advanced search capabilities. The feedback system allows you to view discussion threads in a tree-like

representation, post comments, reply to already posted comments, use stylized comments, and define administrators and moderators.

The DocBook 4 WebHelp with Feedback transformation

To publish DocBook 4 documents to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation. You can customize the out-of-the-box transformation by editing some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. 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, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on;
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server.

Before the transformation starts, enter the documentation product ID and the documentation version. After you run a **DocBook WebHelp with Feedback** transformation, your default browser opens the `instalation.html` file. This file contains information about the output location, system requirements, installation instructions and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

Installation

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Editor plugin demands the following system requirements:

- Apache Web Server running;
- MySQL server running;
- PHP Version $\geq 5.1.6$;
- PHP MySQL Support;
- oXygen 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 the PHP.

Install XAMPP:

- Go to <http://www.apachefriends.org/en/xampp-windows.html> and download XAMPP, for instance for a Windows system;
- Install it in `C:\xampp`;

- From the XAMPP control panel, start "MySQL", and then "Apache";
- Open `http://localhost/xampp/index.php` in your browser to check whether the HTTP server is working.

Create a database for the feedback system and a MySQL user with privileges on that database. The feedback system uses the MySQL user to connect to the database. The username and password for the database administrator are pre-required. For XAMPP, the defaults are `root` for the username and the password is empty.

Use *phpMyAdmin* to create a database:

- Type *localhost* in your browser;
- In the left area, select: *phpMyAdmin*;
- Click *Databases* (in the right frame) and then create a *database*. You can give any name you want to your database, for example *comments*;
- Create a user with connection privileges for this database. In the **SQL** tab, paste the following text:

```
INSERT INTO `mysql`.`user`
(`Host`,`User`,`Password`,`Select_priv`,`Insert_priv`,`Update_priv`,`Delete_priv`,`Create_priv`,
`Drop_priv`,`Reload_priv`,`Shutdown_priv`,`Process_priv`,`File_priv`,`Grant_priv`,`References_priv`,`Index_priv`,`Alter_priv`,
`Show_db_priv`,`Super_priv`,`Create_tmp_table_priv`,`Lock_tables_priv`,`Execute_priv`,`Repl_slave_priv`,`Repl_client_priv`,
`Create_view_priv`,
`Show_view_priv`,`Create_routine_priv`,`Alter_routine_priv`,`Create_user_priv`,`Event_priv`,`Trigger_priv`,
`Create_tablespace_priv`,`ssl_type`,`max_questions`,`max_updates`,`max_connections`,`max_user_connections`,`plugin`,
`authentication_string`) VALUES ('localhost', 'user_name', PASSWORD('user_password'),
'Y','Y','Y','Y','Y','Y','Y','N','N','N',
'N','Y','Y','Y','Y','N','Y','Y','Y','Y','Y','Y','N','Y','Y','Y','', '0', '0', '0',
'0', '', '');
```

- Change the *user_name* and the *user_password* values;
- 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:

- 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`;
- Copy the transformation output folder in the `DOCUMENT_ROOT`;
- Rename it to a relevant name, for example, `webhelp_1`;
- Open `http://localhost/webhelp_1/`. You are redirected to `http://localhost/webhelp_1/install/`;

- Verify that the prerequisites are fulfilled;
- Press **Start Installation**;
- Configure the e-mail addresses, username, and password. Enter the WebHelp system administrator password. This special user is able to moderate new posts and manage WebHelp users;

The address and port of the SMTP mail server that will send the email notifications for the user registration and for the user comments are displayed in the WebHelp installation wizard. They can be modified later in the PHP Runtime Configuration which is usually stored in the `php.ini` file in the XAMPP installation.


You can choose to share comments with other products using the same database. After configuring the database connection, enable the **Display comments from other products** option. In the **Display comments from** section a list with the products sharing the same database is displayed. You can choose one or more products from this list to share comments with.



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 or post as anonymous. Check that the notification emails are delivered to your inbox.

 **Note:** To read debug messages, do 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.

Layout of the Feedback-Enabled WebHelp System

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp, the left frame remaining the same. However, the bottom of the right frame contains a **comments** bar. Select **Log in** from this bar to authenticate as a user of the WebHelp system. In case you do not have a user name, complete the fields in the dialog box that opens to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment whether you are logged in or not. The tabs in the left frame have the same functionality as the Content, Search, and Index tab of the basic WebHelp.

 **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The [WebHelp customization](#) topic contains more details about this.

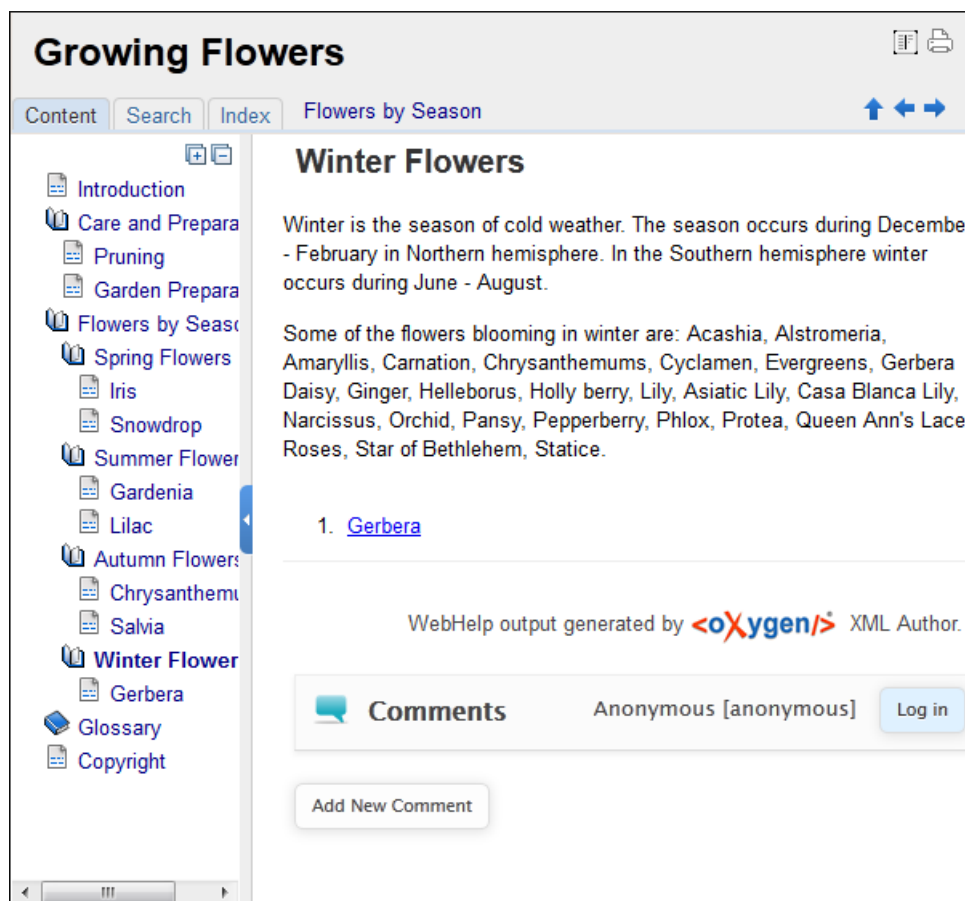


Figure 212: The layout of the Feedback-Enabled WebHelp System

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.

Advanced Customization and Management

As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. To access the administrative page, select **Admin Panel** from the **Comments** bar.

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 213: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp. You can also view the details of each user and search through these details using the **Search User Information** filter.



Note: When you delete a comment, all the replies to that comment are deleted.

To edit the details of a user, click its row and use the **Edit User admin** dialog. In this dialog, you can customize all the information of an user, including is **Status** and **Level**. The following options are available:

- **User Name** - specifies the User Name of the currently edited user;
- **Name** - specifies the name of the currently edited user;
- **Level** - use this field to modify the level of the currently edited user. You can choose from **Admin**, **User**, and **Moderator**;
- **Company** - specifies the company of the selected user;
- **E-mail** - specifies the e-mail address that the currently edited user used to create his account. This is also the address where notifications are sent;
- **Date** - specifies the date when the currently edited user created his account;
- **Web Help Notification** - specifies whether the currently edited user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system;
- **Reply Notification** - specifies whether the currently edited user receives notifications when comments are posted as a reply to a comment left by the user itself;
- **Page Notification** - specifies whether the currently edited user receives notifications when comments are posted on a page where the user posted a comment.;
- **Status** - specifies the status of an user:

- **created** - the currently edited user is created but does not have any rights over the feedback-enabled WebHelp system;
- **validated** - the currently edited user is able to use the feedback-enabled WebHelp system;
- **suspended** - the currently edited user has no rights over the feedback-enabled WebHelp system.

The rights of the users depend on their level as follows:

- **user** - this type of user is able to post comments and receive e-mails when comments are posted anywhere in the documentation, on a single page where he posted a comment, or when a reply to one of his comments is posted;
- **moderator** - apart from the rights of a normal user, this type of user has access to the **Admin Panel**. On the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system;



Note: Comments of version newer than the current version are not displayed.


- **admin** - the administrator has full access to all features of the feedback enabled WebHelp system.

WebHelp Mobile Output Format

To further improve its ability to create online documentation, Oxygen XML Editor plugin offers support to transform DocBook 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, offering table of contents, search capabilities, and index navigation, organized in an intuitive layout.



Figure 214: Mobile WebHelp

To generate a mobile WebHelp system from your DocBook 4 document, go to the **DITA Maps Manager** view, click  **Configure Transformation Scenarios()** and select the **DocBook WebHelp - Mobile** transformation scenario from the **DocBook 5** section. Click **Apply associated**. Once Oxygen XML Editor plugin finishes the transformation process, the output is opened in your default browser automatically.

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 `${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
  SYSTEM "file:///tools/docbook-xsl/common/targetdatabase.dtd" [
<!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
<!ENTITY agtargetsets 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>
    </sitemap>
  </targetset>

```

An example of a target .db file:

```

<!DOCTYPE div
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">
  <ttitle>Administering User Accounts</ttitle>
  <xref>How to administer user accounts</xref>
  <div element="part" href="#d5e4" number="1">
    <ttitle>First Part</ttitle>
    <xref>Part I, "First Part"</xref>
    <div element="chapter" href="#d5e6" number="1">
      <ttitle>Chapter Title</ttitle>
      <xref>Chapter 1, Chapter Title</xref>
      <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
        <ttitle>Section1 Title</ttitle>
        <xref>xreflabel_here</xref>
      </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 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`.

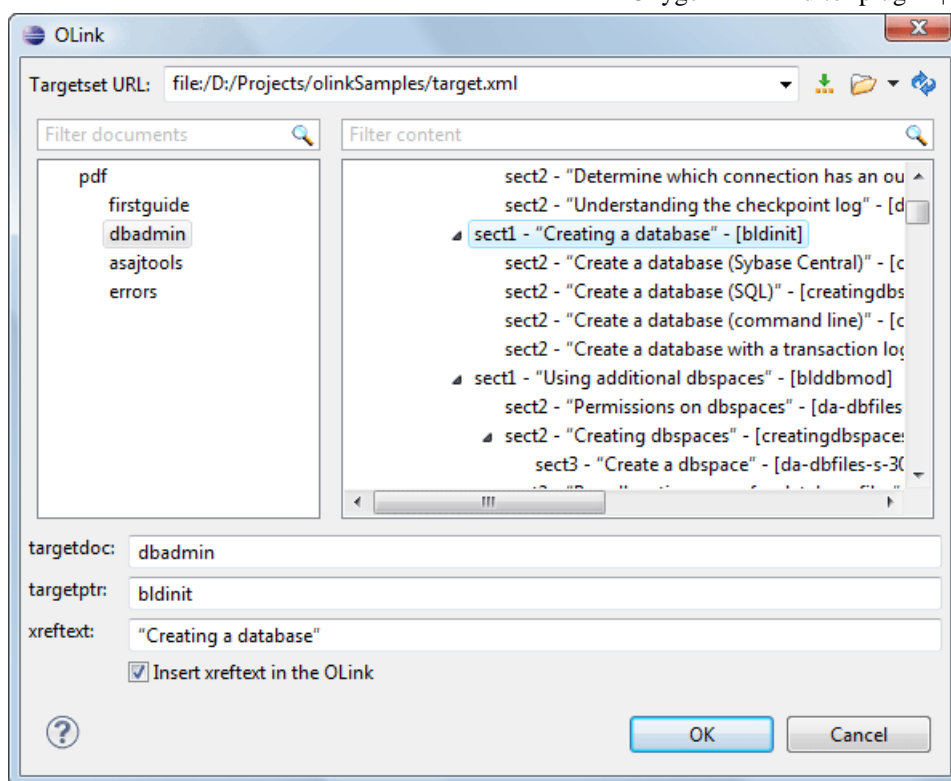


Figure 215: Insert OLink Dialog

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`.

DocBook 5 documents use a Relax NG and Schematron schema located in ``${frameworks}/docbook/5.0/rng/docbookxi.rng`, where ``${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DocBook content is located in ``${frameworks}/docbook/css/docbook.css`.

The XML catalog is stored in ``${frameworks}/docbook/5.0/catalog.xml`.

To watch our video demonstration about editing DocBook documents, go to http://oxygenxml.com/demo/DocBook_Editing_in_Author.html.

DocBook 5 Author Extensions

The DocBook 5 extensions are the same as the *DocBook 4 extensions*.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a DocBook 5 document that is edited in **Author** mode will create a link to the dragged file (the `link` DocBook element) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder

on Mac OS X, etc) will insert an image element (the `inlinemediaobject` DocBook element with an `imagedata` child element) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

DocBook 5 Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp, EPUB, and EPUB 3.

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.


WebHelp Output Format


WebHelp is a form of online help consisting 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.

Oxygen XML Editor plugin allows you to publish DocBook5 documents into a WebHelp format which provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:


- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

 **Note:** In case your documents contain no `indexterm` elements, the **Index** tab is not generated.

 **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The [WebHelp customization](#) topic contains more details about this.

- the right frame 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.

You can use this button , displayed in the **Content** tab, to collapse all the topics presented 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 presents the table of contents in the left side and a section that presents the content of a topic in the right side;

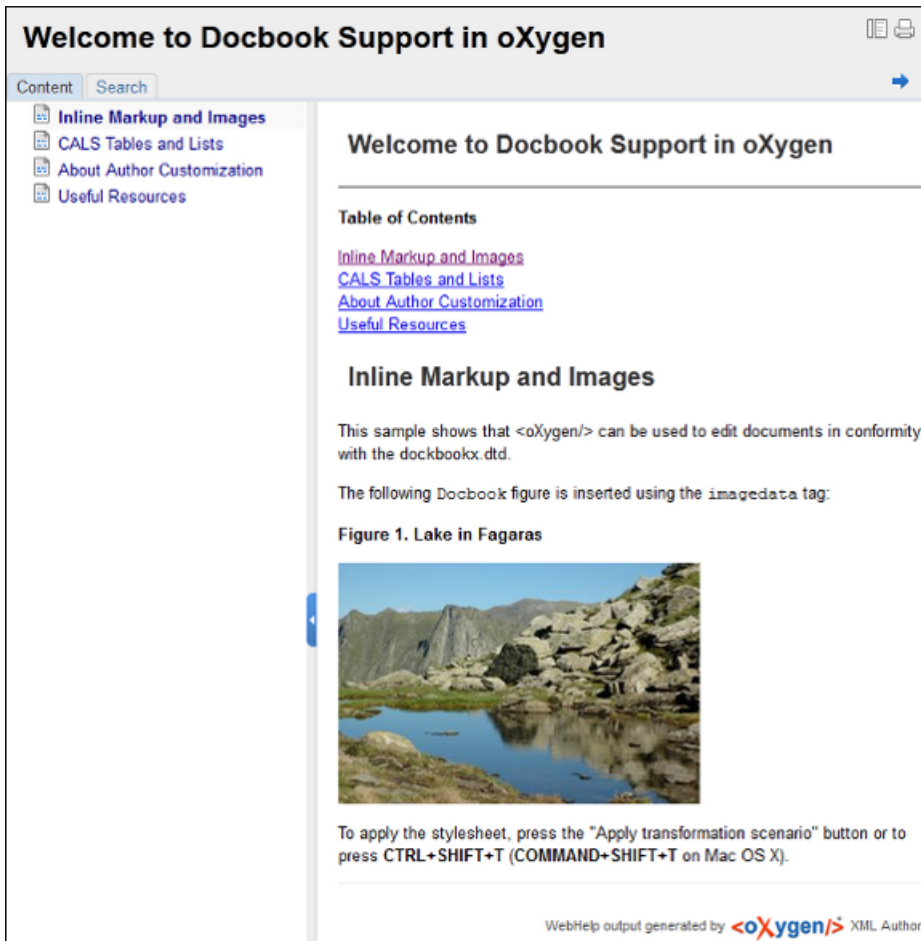


Figure 216: DocBook WebHelp

To publish DocBook 5 to WebHelp, use the **DocBook WebHelp** transformation. To further customize the out-of-the-box transformation, you can edit some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. 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, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on
- `xml.file` - this parameter specifies the path to the DocBook XML file.

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:

- number of keywords found in a single page. The higher the number, the better;
- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

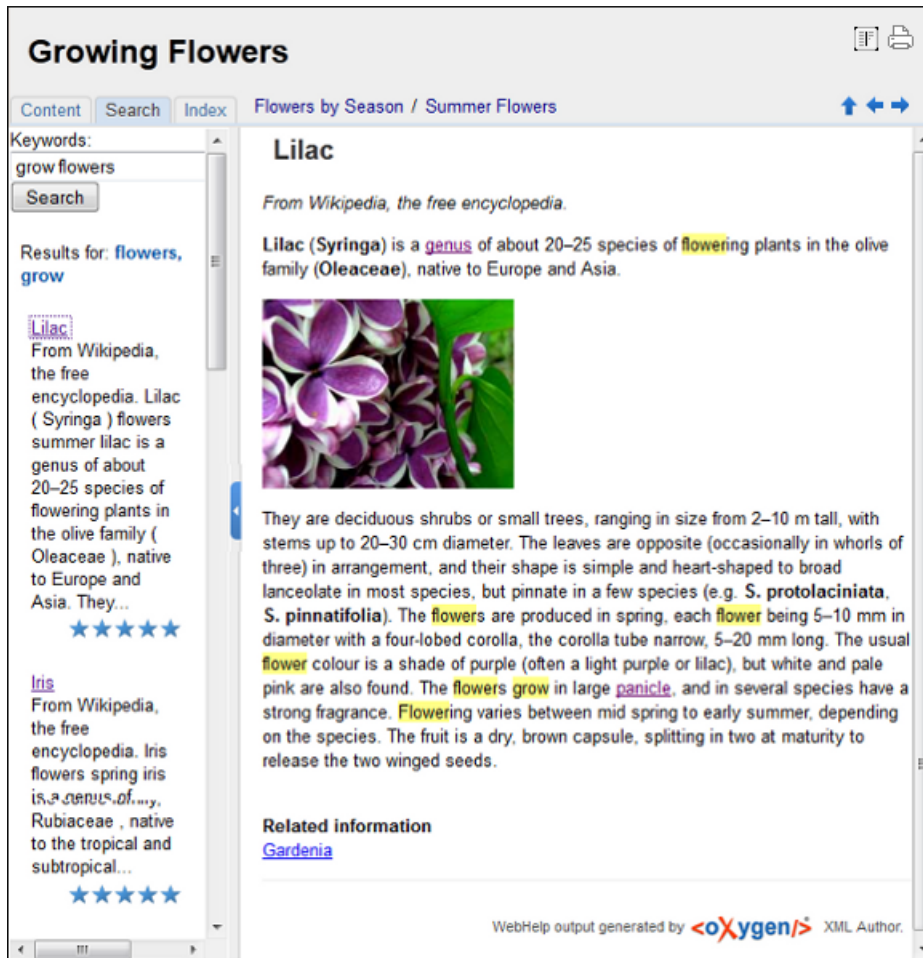



Figure 217: WebHelp Search with Stemming Enabled

 **Note:** This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;
- Opera.


WebHelp with Feedback Output Format


WebHelp is a form of online help consisting 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.

Oxygen XML Editor plugin allows you to publish DocBook5 with Feedback documents into a WebHelp format which provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

 **Note:** In case your documents contain no *indexterm* elements, the **Index** tab is not generated.

 **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

- the right frame 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.

To publish DocBook 5 to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation. To further customize the out-of-the-box transformation, you can edit some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. 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, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server;
- `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;
- `xml.file` - this parameter specifies the path to the DocBook XML file.

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:

- number of keywords found in a single page. The higher the number, the better;
- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"grow flowers"*, returns no results in our case, because it searches for two separate words: *"grow and flowers"* (note the quote signs attached to each word);
- *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 a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

To watch our video demonstration about the feedback-enabled WebHelp system, go to http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.



Note: 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 you to load WebHelp pages in Google Chrome only from a web server.



Note: In case you need to automate the transformation process and use it outside of Oxygen XML Editor plugin, you can use *the Oxygen WebHelp plugin*.

Introduction

Oxygen XML Editor plugin has the ability to transform DocBook 5 documents into feedback-enabled WebHelp systems.

WebHelp is a form of online help consisting 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. It also provides table of contents and advanced search capabilities. The feedback system allows you to view discussion threads in a tree-like representation, post comments, reply to already posted comments, use stylized comments, and define administrators and moderators.

The DocBook 5 WebHelp with Feedback transformation

To publish DocBook 5 documents to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation scenario. You can customize the out-of-the-box transformation by editing some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. 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, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server;
- `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;
- `xml.file` - this parameter specifies the path to the DocBook XML file.

For further information about all the DocBook transformation parameters, go to <http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

Before the transformation starts, enter the documentation product ID and the documentation version. After you run a **DocBook WebHelp with Feedback** transformation, your default browser opens the `instalation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

Installation

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Editor plugin demands the following system requirements:

- Apache Web Server running;
- MySQL server running;

- PHP Version >= 5.1.6;
- PHP MySQL Support;
- oXygen 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 the PHP.

Install XAMPP:

- Go to <http://www.apachefriends.org/en/xampp-windows.html> and download XAMPP, for instance for a Windows system;
- Install it in C:\xampp;
- From the XAMPP control panel, start "MySQL", and then "Apache";
- Open `http://localhost/xampp/index.php` in your browser to check whether the HTTP server is working.

Create a database for the feedback system and a MySQL user with privileges on that database. The feedback system uses the MySQL user to connect to the database. The username and password for the database administrator are pre-required. For XAMPP, the defaults are `root` for the username and the password is empty.

Use *phpMyAdmin* to create a database:

- Type `localhost` in your browser;
- In the left area, select: *phpMyAdmin*;
- Click *Databases* (in the right frame) and then create a *database*. You can give any name you want to your database, for example *comments*;
- Create a user with connection privileges for this database. In the **SQL** tab, paste the following text:

```
INSERT INTO `mysql`.`user`
(`Host`,`User`,`Password`,`Select_priv`,`Insert_priv`,`Update_priv`,`Delete_priv`,`Create_priv`,
`Drop_priv`,`Reload_priv`,`Shutdown_priv`,`Process_priv`,`File_priv`,`Grant_priv`,`References_priv`,`Index_priv`,`Alter_priv`,
`Show_db_priv`,`Super_priv`,`Create_tmp_table_priv`,`Lock_tables_priv`,`Execute_priv`,`Repl_slave_priv`,`Repl_client_priv`,
`Create_view_priv`,
`Show_view_priv`,`Create_routine_priv`,`Alter_routine_priv`,`Create_user_priv`,`Event_priv`,`Trigger_priv`,
`Create_tablespace_priv`,`ssl_type`,`max_questions`,`max_updates`,`max_connections`,`max_user_connections`,`plugin`,
`authentication_string`) VALUES ('localhost', 'user_name', PASSWORD('user_password'),
'Y','Y','Y','Y','Y','Y','Y','N','N','N',
'N','Y','Y','Y','Y','N','Y','Y','Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y','', '0', '0', '0',
'0', '', '');
```

- Change the `user_name` and the `user_password` values;
- 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:

- 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`;
- Copy the transformation output folder in the `DOCUMENT_ROOT`;

- Rename it to a relevant name, for example, `webhelp_1`;
- Open `http://localhost/webhelp_1/`. You are redirected to `http://localhost/webhelp_1/install/`;
 - Verify that the prerequisites are fulfilled;
 - Press **Start Installation**;
 - Configure the e-mail addresses, username, and password. Enter the WebHelp system administrator password. This special user is able to moderate new posts and manage WebHelp users;

The address and port of the SMTP mail server that will send the email notifications for the user registration and for the user comments are displayed in the WebHelp installation wizard. They can be modified later in the PHP Runtime Configuration which is usually stored in the `php.ini` file in the XAMPP installation.

You can choose to share comments with other products using the same database. After configuring the database connection, enable the **Display comments from other products** option. In the **Display comments from** section a list with the products sharing the same database is displayed. You can choose one or more products from this list to share comments with.



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 or post as anonymous. Check that the notification emails are delivered to your inbox.



Note: To read debug messages, do 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.

Layout of the Feedback-Enabled WebHelp System

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp, the left frame remaining the same. However, the bottom of the right frame contains a **comments** bar. Select **Log in** from this bar to authenticate as a user of the WebHelp system. In case you do not have a user name, complete the fields in the dialog box that opens to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment whether you are logged in or not. The tabs in the left frame have the same functionality as the Content, Search, and Index tab of the basic WebHelp.



Note: You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

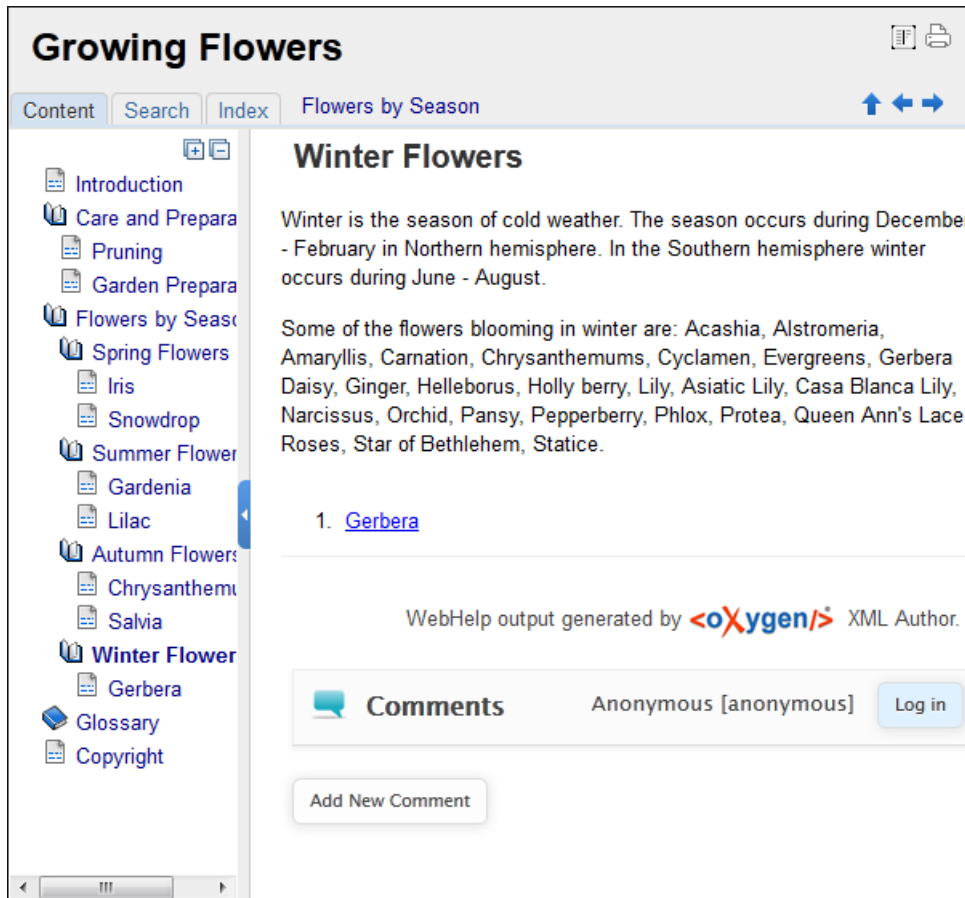



Figure 218: The layout of the Feedback-Enabled WebHelp System

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.


Advanced Customization and Management

As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. To access the administrative page, select **Admin Panel** from the **Comments** bar.

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 219: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp. You can also view the details of each user and search through these details using the **Search User Information** filter.


 **Note:** When you delete a comment, all the replies to that comment are deleted.

To edit the details of a user, click its row and use the **Edit User admin** dialog. In this dialog, you can customize all the information of an user, including is **Status** and **Level**. The following options are available:

- **User Name** - specifies the User Name of the currently edited user;
- **Name** - specifies the name of the currently edited user;
- **Level** - use this field to modify the level of the currently edited user. You can choose from **Admin**, **User**, and **Moderator**;
- **Company** - specifies the company of the selected user;
- **E-mail** - specifies the e-mail address that the currently edited user used to create his account. This is also the address where notifications are sent;
- **Date** - specifies the date when the currently edited user created his account;
- **Web Help Notification** - specifies whether the currently edited user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system;
- **Reply Notification** - specifies whether the currently edited user receives notifications when comments are posted as a reply to a comment left by the user itself;
- **Page Notification** - specifies whether the currently edited user receives notifications when comments are posted on a page where the user posted a comment.;
- **Status** - specifies the status of an user:
 - **created** - the currently edited user is created but does not have any rights over the feedback-enabled WebHelp system;
 - **validated** - the currently edited user is able to use the feedback-enabled WebHelp system;
 - **suspended** - the currently edited user has no rights over the feedback-enabled WebHelp system.

The rights of the users depend on their level as follows:

- **user** - this type of user is able to post comments and receive e-mails when comments are posted anywhere in the documentation, on a single page where he posted a comment, or when a reply to one of his comments is posted;
- **moderator** - apart from the rights of a normal user, this type of user has access to the **Admin Panel**. On the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system;

 **Note:** Comments of version newer than the current version are not displayed.


- **admin** - the administrator has full access to all features of the feedback enabled WebHelp system.

WebHelp Mobile Output format

To further improve its ability to create online documentation, Oxygen XML Editor plugin offers support to transform DocBook 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, offering table of contents, search capabilities, and index navigation, organized in an intuitive layout.



Figure 220: Mobile WebHelp

To generate a mobile WebHelp system from your DocBook 5 document, go to the **DITA Maps Manager** view, click  **Configure Transformation Scenarios()** and select the **DocBook WebHelp - Mobile** transformation scenario from the **DocBook 5** section. Click **Apply associated**. Once Oxygen XML Editor plugin finishes the transformation process, the output is opened in your default browser automatically.

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-install-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>
```

2. Generate a new XSLT stylesheet from the title spec file from the previous step.

Apply `[Oxygen-install-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 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 `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
  SYSTEM "file:///tools/docbook-xsl/common/targetdatabase.dtd" [
<!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
<!ENTITY agtargetes 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">
            &ugtargetes;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargetes;
          </document>
        </dir>
      </dir>
      <dir name="reference">
        <dir name="mailref">
          <document targetdoc="MailReference">
            &reftargetes;
          </document>
        </dir>
      </dir>
    </sitemap>
  </targetset>
```

An example of a target .db file:

```
<!DOCTYPE div
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">
  <ttl>Administering User Accounts</ttl>
  <xrefext>How to administer user accounts</xrefext>
  <div element="part" href="#d5e4" number="I">
    <ttl>First Part</ttl>
    <xrefext>Part I, "First Part"</xrefext>
    <div element="chapter" href="#d5e6" number="1">
      <ttl>Chapter Title</ttl>
      <xrefext>Chapter 1, Chapter Title</xrefext>
      <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
        <ttl>Section1 Title</ttl>
        <xrefext>xreflabel_here</xrefext>
      </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 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`.

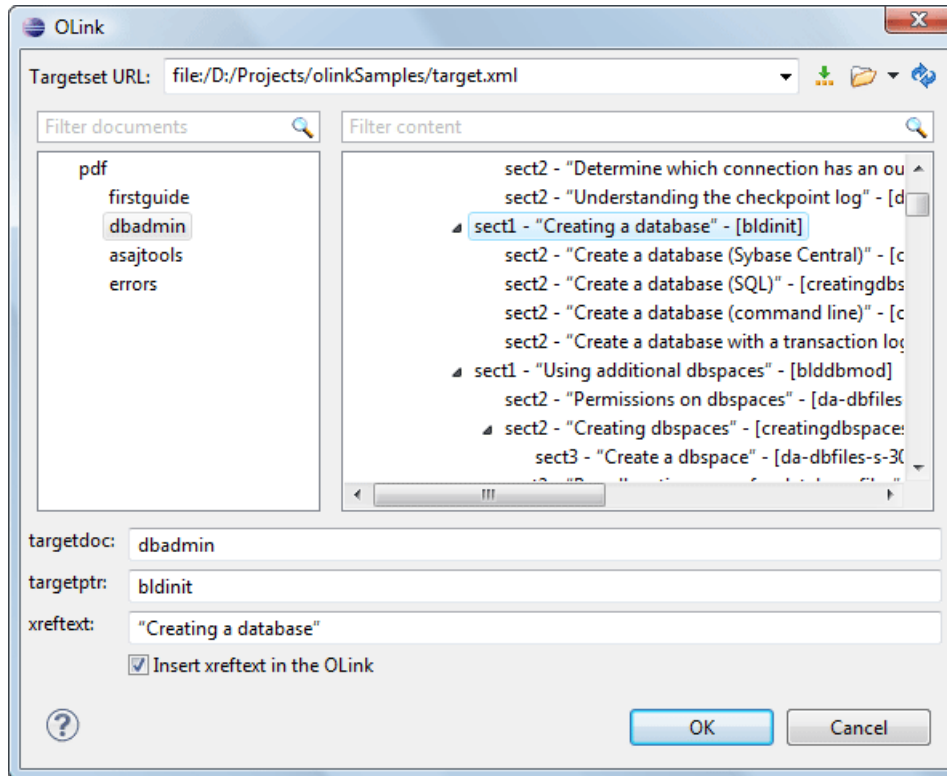


Figure 221: Insert OLink Dialog

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 Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with DocBook `olink`'s.

A file is considered to be a *Targetset* when the root name is `targetset`.

This type of documents use a DTD and schema located in

`${frameworks}/docbook/xsl/common/targetdatabase.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

DocBook Targetset Templates

There is a default template for *Targetset* documents in the `/${frameworks}/docbook/templates/Targetset` folder. It is available when creating *new documents from templates*.

- **Docbook Targetset - Map** - New Targetset Map.

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 different deliverables. The extensibility of DITA permits organizations to define specific information structures and still use standard tools to work with them. Oxygen XML Editor plugin provides schema driven (DTD, RNG, XSD) templates for DITA documents.

A file is considered to be a DITA topic document when either of the following occurs:

- the root element name is one of the following: `concept`, `task`, `reference`, `dita`, `topic`;
- PUBLIC ID of the document is one of the PUBLIC ID's for the elements above;
- the root element of the file has an attribute named `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 Association preferences page* is enabled.

The default schema used for DITA topic documents is located in `/${frameworks}/dita/dtd/ditabase.dtd`, where `/${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DITA content in Author mode is `/${frameworks}/dita/css/dita.css`.

The default XML catalog is `/${frameworks}/dita/catalog.xml`.

DITA Author Extensions

The specific actions for a DITA topic are:

- **B Bold** - surrounds the selected text with a `b` tag;
- **I Italic** - surrounds the selected text with an `i` tag;
- **U Underline** - surrounds the selected text with a `u` tag;



Note:

Bold, **Italic**, and **Underline** are toggle actions.

For all of the above actions, if there is no selection in the document, then a new specific tag is inserted. These actions are available in any document context.

- **Style Guide** - opens the **DITA Style Guide Best Practices for Authors** in your browser and presents the description of the element at the caret position. This action is available in the contextual menu of the editing area (when editing DITA documents), in the **DITA** menu, and in the documentation tip 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;
- **Cross Reference** - inserts an `xref` element with the value of the `format` attribute set to `dita`. The target of the `xref` element 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.

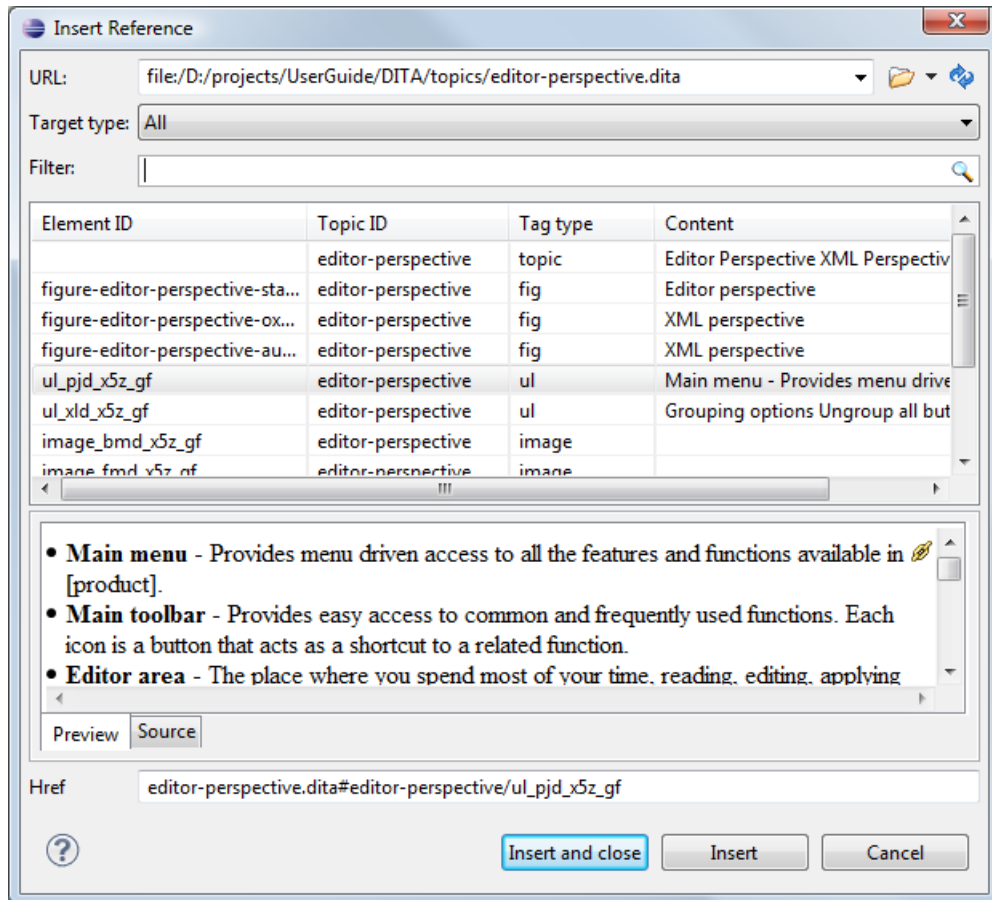



Figure 222: Insert a Cross Reference in a DITA Document

 **Note:** The **Insert Reference** dialog is not modal. The dialog 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 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 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;

 **Note:** The actions for inserting references described above are grouped in the **DITA > Link** menu.

- **Paste as content reference** (available on the contextual menu of Author editor for any topic file) - 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 as content key reference** - allows you to indirectly refer 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:
 - set the `id` attribute of the element holding the content you want to refer;
 - open the DITA Map in the **DITA Maps Manager** view;
 - right click the topic that holds the content you want to refer, select **Edit Properties**, and enter a value in the **Keys** field;
 - make sure the DITA Map remains open in the DITA MAPS Manger View.
- **paste as link** (available on the contextual menu of Author editor for any topic file) - inserts a `link` element or an `xref` one (depending on the location of the paste) 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/href` will point to this ID value;
- **Paste as link (keyref)** - inserts a link to the element that you want to refer. To use this action, do the following:
 - set the `id` attribute of the element that you want to refer;
 - open the DITA Map in the **DITA Maps Manager** view;
 - right click the topic that holds the content you want to refer, select **Edit Properties**, and enter a value in the **Keys** field;
 - make sure the DITA Map remains open in the DITA MAPS Manger View.
- **§ Insert Section / Step** - 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 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 image reference** - *inserts an image reference* at the caret position. Depending on the current context, an image-type DITA element is inserted. Also you can use this action to *refer MathML files*;
-  **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 plugin *displays the referred 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 plugin*. If the URI's can be resolved the referred 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**.

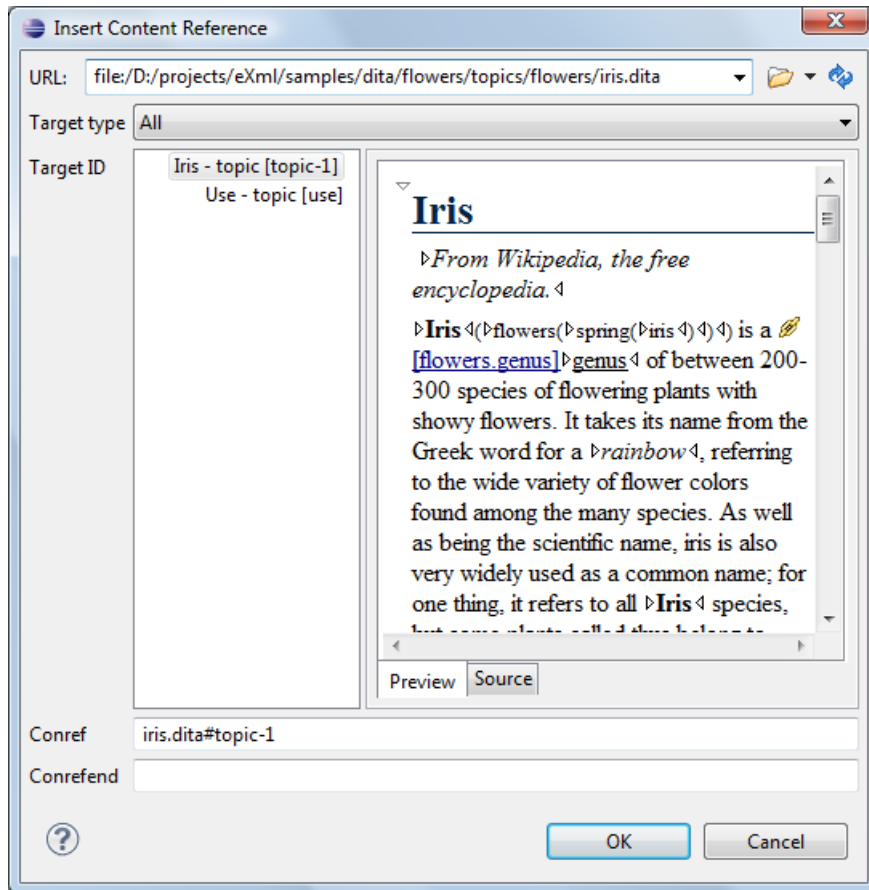



Figure 223: Insert Content Reference Dialog

 **Note:** The **Insert Content Reference** dialog is not modal. The dialog 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 referred (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 **OK** 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 plugin 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 plugin *displays the key referred 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**.

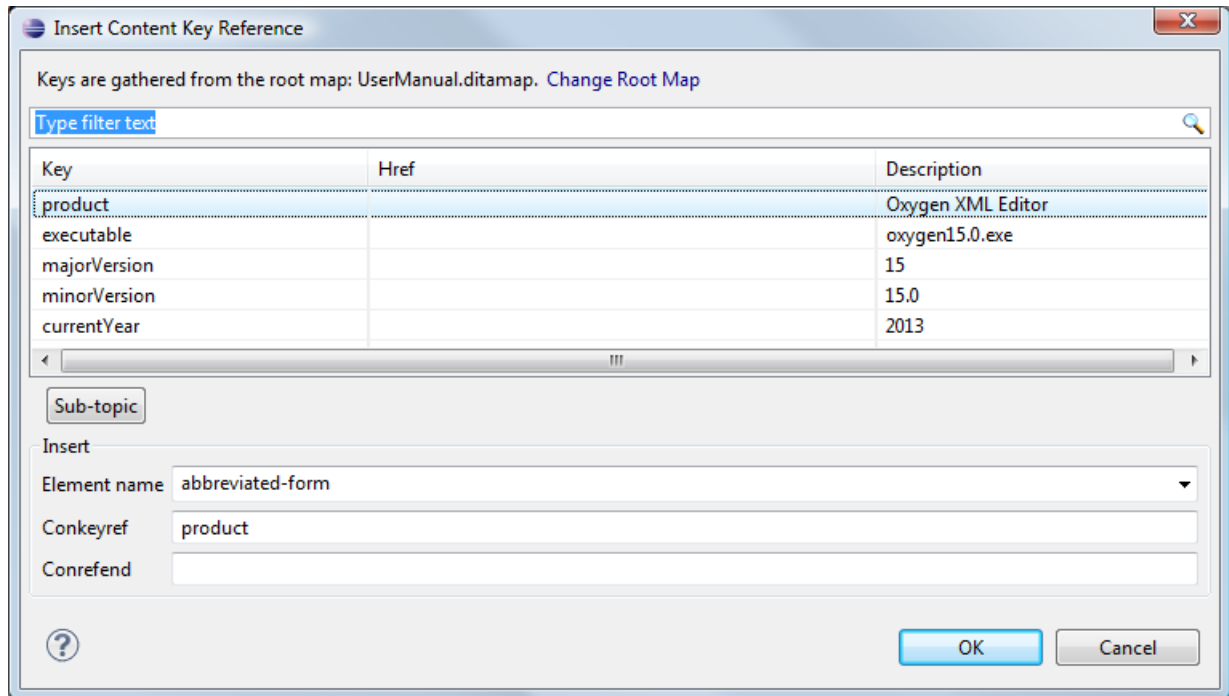






Figure 224: Insert Content Key Reference Dialog






















 **Note:** The **Insert Content Key Reference** dialog is not modal. The dialog 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 plugin will present the entire referenced range as read-only content.

 **Important:** All keys which are presented in the dialog 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.

- **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. ;
- **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](#);
-  **Paste as Content Reference** - pastes the content of the clipboard as a content reference. Note that the copied element must have the `id` attribute set;
-  **Paste as Link** - pastes the content of the clipboard as a link. Note that the copied element must have the `id` attribute set;

-  **Insert an ordered list at the caret position** - inserts an ordered list. A child list item is also inserted automatically by default;
 -  **Insert an unordered list at the caret position** - inserts an itemized list. A child list item is also inserted automatically by default;
 -  **Insert a step or list Item** - inserts a new list item in any of the above list types.
 -  **Insert Table** - opens a dialog 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 plugin 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;
 -  **Insert Row Above** - inserts a row above the current one;
 - **Insert Row Below** - inserts a row below the current one;
 -  **Insert Column Before** - inserts a column before the current one;
 - **Insert Column After** - inserts a column after the current one;
 -  **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 plugin deletes the source row is 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 plugin 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 plugin 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 plugin 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 plugin decreases the column span of the source cell with one.
- The following options are available for customizing the frame of a table:
 - none - the table has no frame;
 - all - the table has top, bottom and side borders;

- `top` - the table has a top border;
- `topbot` - the table top and bottom borders;
- `bottom` - the table has a bottom border;
- `sides` - the table has side border;
- `-dita-use-conref-target` - it indicates that when the element uses *conref* to pull in content, the attribute with a value of **-dita-use-conref-target** should also be pulled in from the target;
- `unspecified` - if you select this option, when you insert the table, the frame attribute is not inserted.



Note: DITA supports the CALS table model similar with DocBook document type in addition to the `simpletable` element specific for DITA.



Caution: Column specifications are required for table actions to work properly.

- **Generate IDs** - Available in the contextual menu, this action allows you to generate an unique ID for the element at caret position. If the element already has an ID set, it is preserved.

Further options are offered in the **ID Options** dialog, available in the document type specific main menu. The configurable ID value pattern can accept most of the application supported *editor variables*. You can also specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled.

If you want to keep an already set element ID's when copying content in the same document, make sure the **Remove ID's 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 (Meta on Mac OS) + Shift + G** and can be changed in the **DITA Topic** document type.

All actions described above are available in the contextual menu, the **DITA** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a DITA topic document that is edited in Author mode will create a link to the dragged file (the `xref` DITA element with the `href` attribute) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `image` DITA element with the `href` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

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 1.6.1;
- **DITA PDF** - Transforms a DITA topic to PDF using the DITA Open Toolkit 1.6.1 and the Apache FOP engine.

DITA Templates

The default templates available for DITA topics are stored in `${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, writers, and publishers to plan, develop, and deliver content.

A file is considered to be a DITA map document when either of the following occurs:

- root element name is one of the following: `map`, `bookmap`;
- 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 schema used for DITA map documents is located in `#{frameworks}/dita/DITA-OT/dtd/map.dtd`, where `#{frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.








The CSS file used for rendering DITA content is located in `#{frameworks}/dita/css/dita.css`.

The default XML catalog is stored in `#{frameworks}/dita/catalog.xml`.

DITA Map Author Extensions

Specific actions for DITA map documents are:

-  **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 reference at the caret position.
-  **Insert Table** - Opens a dialog that allows you to configure the relationship table to be inserted. The dialog 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.
-  **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.

All actions described above are available in the contextual menu, the **DITA** submenu of the main menu and in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a DITA map document that is edited in Author mode will create a link to the dragged file (a `topicref` element, a chapter one, a part one, 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 DITA-OT/plugins directory.
- **DITA Map Metrics Report** - Use this transformation scenario if you want to generate a DITA Map statistics report containing information like:
 - 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 like number of containing maps, bookmaps, or topics.

Many more output formats are available by clicking the **New** button. The transformation process relies on DITA Open Toolkit 1.6.1.

WebHelp Output Format

WebHelp is a form of online help consisting 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.

Oxygen XML Editor plugin allows you to publish a DITA Map into a WebHelp format that provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;



Note: In case your documents contain no *indexterm* elements, the **Index** tab is not generated.




Note: You can choose to enhance the appearance of the selected item in the table of contents. The [WebHelp customization](#) topic contains more details about this.

- the right frame 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 this button , displayed in the **Content** tab, to collapse all the topics presented 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 presents the table of contents in the left side and a section that presents the content of a topic in the right side;

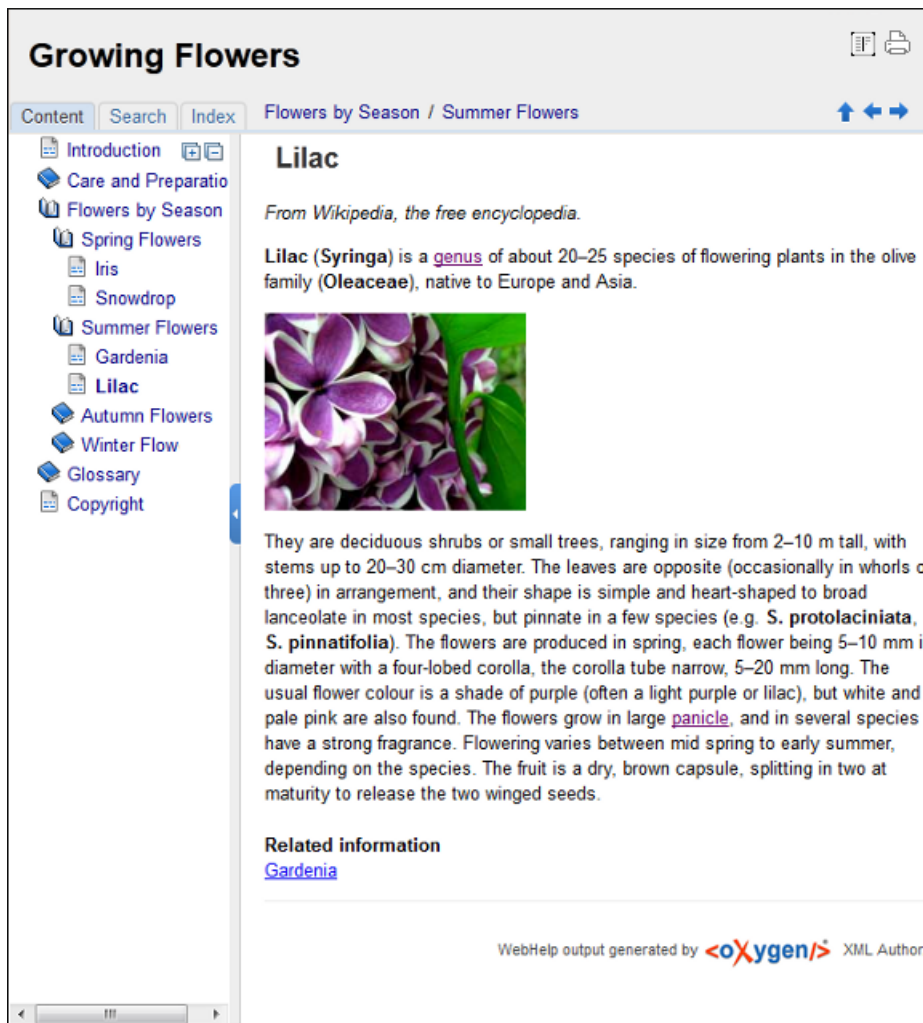


Figure 225: WebHelp Output

To publish the DITA map to WebHelp, you can use the **DITA Map WebHelp** transformation. The [WebHelp Customization](#) topic describes each parameter which can be configured in order to customize the output.

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:

- number of keywords found in a single page. The higher the number, the better;

- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

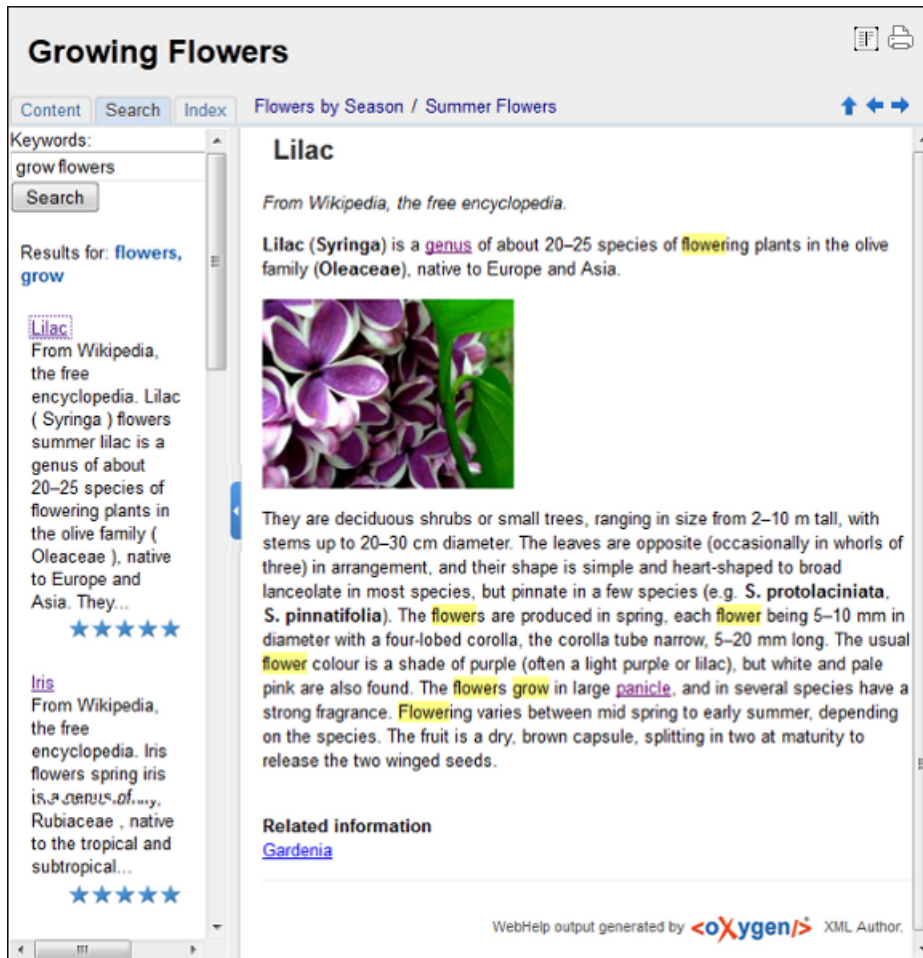



Figure 226: WebHelp Search with Stemming Enabled

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like "*grow flowers*", returns no results in our case, because it searches for two separate words: "*grow* and *flowers*" (note the quote signs attached to each word);
- *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 a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

 **Note:** 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 you to load WebHelp pages in Google Chrome only from a web server.

WebHelp Customizations

This section takes you through the customizations that you can make to the output of your WebHelp transformation. You are able 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. Also, an additional list of WebHelp related parameters is presented.

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 runs.

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` file. You can find this file at **oxygen-webhelp > resources > skins > desktop**. Open the `toc.css` file, find the `menuItemSelected` class and change the value of the `background` property.



Note: Also, you can overwrite the same value from your own CSS.

Adding a Logo Image in the Title Area

You are able to customize the title of your WebHelp output, 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 which you place in the `[Oxygen-install-dir]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img` directory so 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, set in the WebHelp transformation scenario using the `args.css` parameter.

Adding Custom Headers and Footers

In the transformation scenario, you can use the `args.hdr` and `args.ftr` parameters to point to resources which contain your custom HTML `<div>` blocks. These are included in the header and footer of each generated topic.

WebHelp Additional Parameters

The following additional WebHelp related parameters can be configured in the transformation scenario:

- `webhelp.copyright` - add a small copyright text which appears at the end of the table of contents;
- You can edit the `args.hide.parent.link` parameter to hide the **Parent**, **Next**, and **Previous** links;
- `args.xhtml.toc` - name of the table of contents file. Default setting is `toc.html`;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. Default setting is `false`;

- `clean.output` - deletes all files from the output folder before the transformation is performed. Only the `no` and `yes` values are valid. The default value is `no`.

How to Localize WebHelp Output

Static labels used in the WebHelp output are kept in translation files in the `DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\localization` folder. The DITA-OT folder is by default `[oxygen installation folder]/frameworks/dita/DITA-OT`, or 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 (for simplicity sake, let us suppose you want to localize the WebHelp interface into Canadian French.):

1. Look for the `strings-fr-ca.xml` file in `DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\localization`. 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. 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 the interface into (in our case, it is `fr-ca`).
4. Run the transformation scenario to produce the WebHelp output.

WebHelp with Feedback Output Format

This section presents the Feedback-Enabled WebHelp systems support.

Introduction

Oxygen XML Editor plugin offers support to transform DITA documents into feedback-enabled WebHelp systems.

WebHelp is a form of online help consisting 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. It also provides table of contents and advanced search capabilities. The feedback system allows you to view discussion threads in a tree-like representation, post comments, reply to already posted comments, use stylized comments, and define administrators and moderators.



Note: This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;
- Opera.


The DITA map WebHelp with Feedback transformation

To publish a DITA map to WebHelp with Feedback, use the **DITA Map WebHelp with Feedback** transformation. You can customize the out-of-the-box transformation by editing some of its parameters:


- `args.xhtml.toc` - name of the table of contents file. Default setting is `toc.html`;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. Default setting is `false`;
- `clean.output` - deletes all files from the output folder before the transformation is performed. Only the `no` and `yes` values are valid. The default value is `no`.

Before the transformation starts, enter the documentation product ID and the documentation version. After you run a **DITA Map WebHelp with Feedback** transformation, your default browser opens the `instalation.html` file. This file contains information about the output location, system requirements, installation instructions and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

 **Note:** This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;
- Opera.

 **Note:** In case you need to automate the transformation process and use it outside of Oxygen XML Editor plugin, you can use [the Oxygen WebHelp plugin](#).


Installation

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Editor plugin demands the following system requirements:

- Apache Web Server running;
- MySQL server running;
- PHP Version >= 5.1.6;
- PHP MySQL Support;
- oXygen 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 the PHP.

Install XAMPP:

- Go to <http://www.apachefriends.org/en/xampp-windows.html> and download XAMPP, for instance for a Windows system;
- Install it in C:\xampp;
- From the XAMPP control panel, start "MySQL", and then "Apache";
- Open `http://localhost/xampp/index.php` in your browser to check whether the HTTP server is working.

Create a database for the feedback system and a MySQL user with privileges on that database. The feedback system uses the MySQL user to connect to the database. The username and password for the database administrator are pre-required. For XAMPP, the defaults are `root` for the username and the password is empty.

Use *phpMyAdmin* to create a database:

- Type `localhost` in your browser;
- In the left area, select: *phpMyAdmin*;

- Click *Databases* (in the right frame) and then create a *database*. You can give any name you want to your database, for example *comments*;
- Create a user with connection privileges for this database. In the **SQL** tab, paste the following text:

```
INSERT INTO `mysql`.`user`
(`Host`,`User`,`Password`,`Select_priv`,`Insert_priv`,`Update_priv`,`Delete_priv`,`Create_priv`,
`Drop_priv`,`Reload_priv`,`Shutdown_priv`,`Process_priv`,`File_priv`,`Grant_priv`,`References_priv`,`Index_priv`,`Alter_priv`,
`Show_db_priv`,`Super_priv`,`Create_tmp_table_priv`,`Lock_tables_priv`,`Execute_priv`,`Repl_slave_priv`,`Repl_client_priv`,
`Create_view_priv`,
`Show_view_priv`,`Create_routine_priv`,`Alter_routine_priv`,`Create_user_priv`,`Event_priv`,`Trigger_priv`,
`Create_tablespace_priv`,`ssl_type`,`max_questions`,`max_updates`,`max_connections`,`max_user_connections`,`plugin`,
`authentication_string`) VALUES ('localhost','user_name',PASSWORD('user_password'),
'Y','Y','Y','Y','Y','Y','Y','N','N','N',
'N','Y','Y','Y','Y','N','Y','Y','Y','Y','Y','Y','N','Y','Y','Y','',
'0','0','0',
'0','','');
```

- Change the *user_name* and the *user_password* values;
- 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:

- 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;
- Copy the transformation output folder in the DOCUMENT_ROOT;
- Rename it to a relevant name, for example, webhelp_1;
- Open http://localhost/webhelp_1/. You are redirected to http://localhost/webhelp_1/install/;
- Verify that the prerequisites are fulfilled;
- Press **Start Installation**;
- Configure the e-mail addresses, username, and password. Enter the WebHelp system administrator password. This special user is able to moderate new posts and manage WebHelp users;

The address and port of the SMTP mail server that will send the email notifications for the user registration and for the user comments are displayed in the WebHelp installation wizard. They can be modified later in the PHP Runtime Configuration which is usually stored in the `php.ini` file in the XAMPP installation.

You can choose to share comments with other products using the same database. After configuring the database connection, enable the **Display comments from other products** option. In the **Display comments from** section a list with the products sharing the same database is displayed. You can choose one or more products from this list to share comments with.



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 or post as anonymous. Check that the notification emails are delivered to your inbox.




Note: To read debug messages, do 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.

Layout of the Feedback-Enabled WebHelp System

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp, the left frame remaining the same. However, the bottom of the right frame contains a **comments** bar. Select **Log in** from this bar to authenticate as a user of the WebHelp system. In case you do not have a user name, complete the fields in the dialog box that opens to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment whether you are logged in or not. The tabs in the left frame have the same functionality as the Content, Search, and Index tab of the basic WebHelp.

 **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

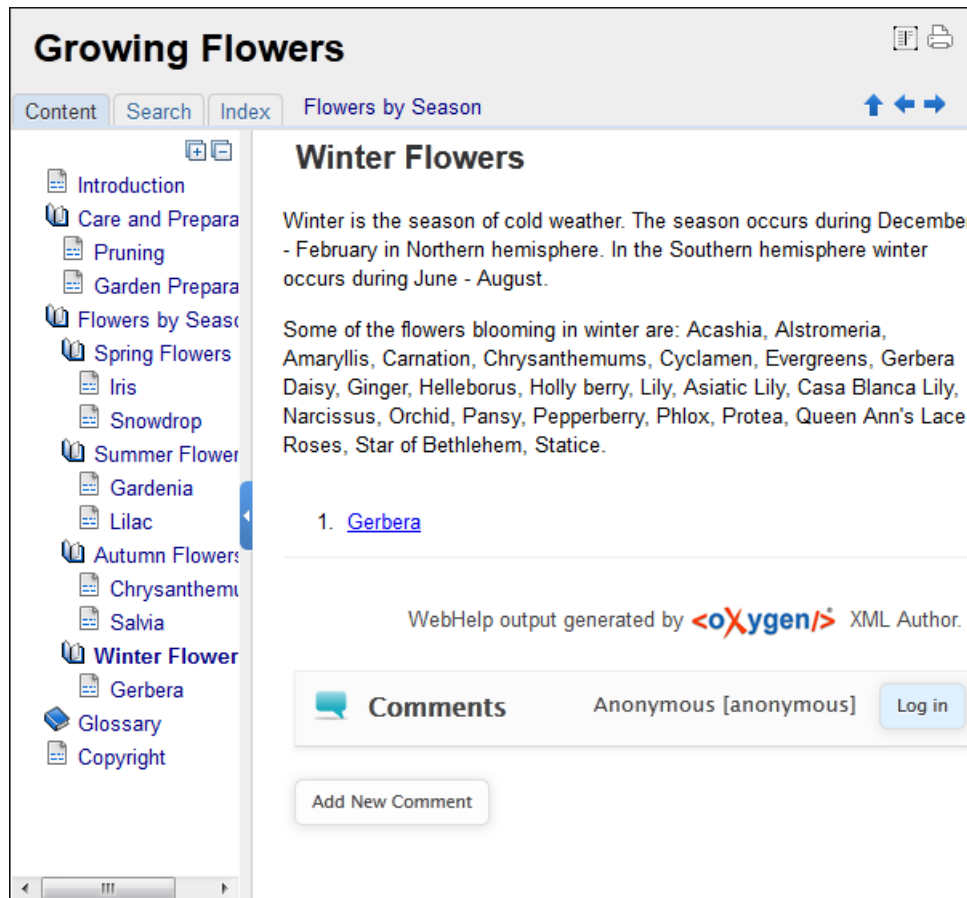


Figure 227: The layout of the Feedback-Enabled WebHelp System

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.

Advanced Customization and Management

As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. To access the administrative page, select **Admin Panel** from the **Comments** bar.

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 228: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp. You can also view the details of each user and search through these details using the **Search User Information** filter.



Note: When you delete a comment, all the replies to that comment are deleted.

To edit the details of a user, click its row and use the **Edit User admin** dialog. In this dialog, you can customize all the information of an user, including is **Status** and **Level**. The following options are available:

- **User Name** - specifies the User Name of the currently edited user;
- **Name** - specifies the name of the currently edited user;
- **Level** - use this field to modify the level of the currently edited user. You can choose from **Admin**, **User**, and **Moderator**;
- **Company** - specifies the company of the selected user;
- **E-mail** - specifies the e-mail address that the currently edited user used to create his account. This is also the address where notifications are sent;
- **Date** - specifies the date when the currently edited user created his account;
- **Web Help Notification** - specifies whether the currently edited user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system;
- **Reply Notification** - specifies whether the currently edited user receives notifications when comments are posted as a reply to a comment left by the user itself;
- **Page Notification** - specifies whether the currently edited user receives notifications when comments are posted on a page where the user posted a comment.;
- **Status** - specifies the status of an user:
 - **created** - the currently edited user is created but does not have any rights over the feedback-enabled WebHelp system;
 - **validated** - the currently edited user is able to use the feedback-enabled WebHelp system;
 - **suspended** - the currently edited user has no rights over the feedback-enabled WebHelp system.

The rights of the users depend on their level as follows:

- **user** - this type of user is able to post comments and receive e-mails when comments are posted anywhere in the documentation, on a single page where he posted a comment, or when a reply to one of his comments is posted;

- **moderator** - apart from the rights of a normal user, this type of user has access to the **Admin Panel**. On the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system;



Note: Comments of version newer than the current version are not displayed.


- **admin** - the administrator has full access to all features of the feedback enabled WebHelp system.

Mobile WebHelp Output Format

To further improve its ability to create online documentation, Oxygen XML Editor plugin offers support to transform 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, offering table of contents, search capabilities, and index navigation, organized in an intuitive layout.



Figure 229: Mobile WebHelp

To generate a mobile WebHelp system from your DITA MAP, go to the **DITA Maps Manager** view, click  **Configure Transformation Scenarios()** and select the **DITA Map WebHelp - Mobile** transformation scenario. Click **Apply associated**. Once Oxygen XML Editor plugin finishes the transformation process, the output is opened in your default browser automatically.

DITA Map Templates

The default templates available for DITA maps are stored in `${frameworks}/dita/templates/map` folder. They can be used for easily creating DITA map and bookmark files.

Here are some of the DITA Map templates available when creating *new documents from templates*:

- **DITA Map - Bookmark** - New DITA Bookmark;
- **DITA Map - Map** - New DITA Map;
- **DITA Map - Learning Map** - New DITA learning and training content specialization map;
- **DITA Map - Learning Bookmark** - New DITA learning and training content specialization bookmark;
- **DITA Map - Eclipse Map** - New DITA learning and training content specialization bookmark.

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, or 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 a `html`.

The schema used for these documents is located in `/${frameworks}/xhtml/dtd/xhtml1-strict.dtd`, where `/${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The default CSS options for the XHTML document type are set to merge the CSSs specified in the document with the CSSs defined in the XHTML document type.

The CSS file used for rendering XHTML content is located in `/${frameworks}/xhtml/css/xhtml.css`.

There are three default catalogs for XHTML document type:

- `/${frameworks}/xhtml/dtd/xhtmlcatalog.xml`;
- `/${frameworks}/xhtml11/dtd/xhtmlcatalog.xml`;
- `/${frameworks}/xhtml11/schema/xhtmlcatalog.xml`.

XHTML Author Extensions

Specific actions are:








- **B Bold** - changes the style of the selected text to `bold` by surrounding it with `b` tag;
- **I Italic** - changes the style of the selected text to `italic` by surrounding it with `i` tag;
- **U Underline** - changes the style of the selected text to `underline` by surrounding it with `u` tag.















Note:





Bold, **Italic**, and **Underline** are toggle actions.

For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

- **H Headings** - groups actions for inserting `h1`, `h2`, `h3`, `h4`, `h5`, `h6` elements;
-  **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;
 -  **Insert an ordered list at the caret position** - inserts an ordered list. A child list item is also inserted automatically by default;
 -  **Insert an unordered list at the caret position** - inserts an itemized list. A child list item is also inserted automatically by default;
 -  **Insert a step or list item** - inserts a new list item in any of the above list types.
-  **Insert Definition List** - inserts a definition list (`dl` element) with one list item (a `dt` child element and a `dd` child element);
 -  **Insert Table** - opens a dialog 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 plugin 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;
-  **Insert Row Above** - inserts a row above the current one;
- **Insert Row Below** - inserts a row below the current one;
-  **Insert Column Before** - inserts a column before the current one;
- **Insert Column After** - inserts a column after the current one;
-  **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 plugin deletes the source row is 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 plugin 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 plugin 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 plugin 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 plugin decreases the column span of the source cell with one.

All actions described above are available in the contextual menu, the **XHTML** submenu of the main menu and in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a DITA topic document that is edited in Author mode will create a link to the dragged file (the `xref` DITA element with the `href` attribute) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `image` DITA element with the `href` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** 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 `${frameworksDir}/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 creating 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 either of the following occurs:

- the file extension is `.odd`
- the document's namespace is `http://www.tei-c.org/ns/1.0`

The schema used for these documents is located in

`${frameworks}/tei/xml/tei/custom/schema/relaxng/brown_odds.rng`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin installation directory.

The CSS file used for rendering TEI ODD content is located in

`${frameworks}/tei/xml/tei/css/tei_oxygen_odd.css`.

There are two default catalogs for TEI ODD document type:

- `${frameworks}/tei/xml/tei/custom/schema/catalog.xml`
- `${frameworks}/tei/xml/tei/schema/catalog.xml`

To watch our video demonstration about TEI editing, go to http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html.

TEI ODD Author Extensions

The specific actions for TEI ODD documents are:























- **B Bold** - changes the style of the selected text to bold by surrounding it with `hi` tag and setting the `rend` attribute to bold;
- **I Italic** - changes the style of the selected text to italic by surrounding it with `hi` tag and setting the `rend` attribute to italic;
- **U Underline** - changes the style of the selected text to underline by surrounding it with `hi` tag and setting the `rend` attribute to ul;




Note:

Bold, **Italic**, and **Underline** are toggle actions.

For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

-  **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 an ordered list at the caret position** - inserts an ordered list. A child list item is also inserted automatically by default;
 -  **Insert an unordered list at the caret position** - inserts an itemized list. A child list item is also inserted automatically by default;
 -  **Insert a step or list Item** - inserts a new list item in any of the above list types.
 -  **Insert Table** - opens a dialog 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 plugin 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;
 -  **Insert Row Above** - inserts a row above the current one;
 - **Insert Row Below** - inserts a row below the current one;
 -  **Insert Column Before** - inserts a column before the current one;
 - **Insert Column After** - inserts a column after the current one;
 -  **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 plugin deletes the source row is 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 plugin 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 plugin 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 plugin 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 plugin decreases the column span of the source cell with one.
- Generate IDs** - Available in the contextual menu, this action allows you to generate an unique ID for the element at caret position. If the element already has an ID set, it is preserved. Further options are offered in the **ID Options** dialog, available in the **TEI ODD** main menu. Here you can specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled. If you want to keep already set element id's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.

All actions described above are available in the contextual menu, the **TEI ODD** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a **TEI ODD** document that is edited in Author mode will create 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 `${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 either of the following occurs:

- the root's local name is `TEI . 2`
- the document's public id is `-//TEI P4`

The DTD schema used for these documents is located in `${frameworks}/tei/tei2xml.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering TEI P4 content is located in
`${frameworks}/tei/xml/tei/css/tei_oxygen.css`.

There are two default catalogs for TEI P4 document type:

- `${frameworks}/tei/xml/teip4/schema/dtd/catalog.xml`
- `${frameworks}/tei/xml/teip4/custom/schema/dtd/catalog.xml`

To watch our video demonstration about TEI editing, go to http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html.

TEI P4 Author Extensions

The specific actions for TEI P4 documents are:












- **B Bold** - changes the style of the selected text to bold by surrounding it with `hi` tag and setting the `rend` attribute to bold;
- **I Italic** - changes the style of the selected text to italic by surrounding it with `hi` tag and setting the `rend` attribute to italic;
- **U Underline** - changes the style of the selected text to underline by surrounding it with `hi` tag and setting the `rend` attribute to ul;










Note:





Bold, **Italic**, and **Underline** are toggle actions.

For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

- **Browse reference manual** - opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position;
- **§ 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 an ordered list at the caret position** - inserts an ordered list. A child list item is also inserted automatically by default;
 -  **Insert an unordered list at the caret position** - inserts an itemized list. A child list item is also inserted automatically by default;
 -  **Insert a step or list Item** - inserts a new list item in any of the above list types.
-  **Insert Table** - opens a dialog 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 plugin 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;
-  **Insert Row Above** - inserts a row above the current one;
- **Insert Row Below** - inserts a row below the current one;

-  **Insert Column Before** - inserts a column before the current one;
- **Insert Column After** - inserts a column after the current one;
-  **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 plugin deletes the source row in 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 plugin 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 plugin 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 plugin 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 plugin decreases the column span of the source cell with one.
- **Generate IDs** - Available in the contextual menu, this action allows you to generate a unique ID for the element at caret position. If the element already has an ID set, it is preserved.

Further options are offered in the **ID Options** dialog, available in the document type specific main menu. The configurable ID value pattern can accept most of the application supported *editor variables*. You can also specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled.

If you want to keep an already set element ID's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.

All actions described above are available in the contextual menu, the **TEI P4** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a TEI P4 document that is edited in Author mode 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 `${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.
 - a) Set the Oxygen `frameworks` folder to the `oxygen/frameworks` subfolder of the folder of the SVN working copy.

Go to menu **Options > Preferences > Global** and set the path of the SVN working copy in the option **Use custom frameworks**.
 - b) Edit the TEI framework in the **Preferences** dialog.

Go to **Options > Preferences > Document Type Association/Locations** and select **Custom**.
 - c) Close the **Preferences** dialog with the **OK** button.

If the dialog is closed with the **Cancel** button, the modifications are not saved on disk.
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 install folder. After restarting the Oxygen application the new TEI framework will appear in the **Options > Preferences > Document Type Association**.

The TEI P5 Document Type

The TEI P5 document type is similar with the TEI P4 one, with the following exceptions:

- A file is considered to be a TEI P5 document when the namespace is `http://www.tei-c.org/ns/1.0`.
- The schema is located in `${frameworks}/tei/xml/tei/custom/schema/relaxng/tei_allPlus.rng`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.
- A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `graphic` DITA element with the `url` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

To watch our video demonstration about TEI editing, go to http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html.

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 `${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.
 - a) Set the Oxygen `frameworks` folder to the `oxygen/frameworks` subfolder of the folder of the SVN working copy.
Go to menu **Options > Preferences > Global** and set the path of the SVN working copy in the option **Use custom frameworks**.
 - b) Edit the TEI framework in the **Preferences** dialog.
Go to **Options > Preferences > Document Type Association/Locations** and select **Custom**.
 - c) Close the **Preferences** dialog with the **OK** button.
If the dialog is closed with the **Cancel** button, the modifications are not saved on disk.
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 install folder. After restarting the Oxygen application the new TEI framework will appear in the **Options > Preferences > Document Type Association**.

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 plugin.

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 plugin `frameworks` folder.

The EPUB Document Type

Three distinct frameworks support 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 conforming to the Open Publication Structure(OPS) standard including metadata, reading order and navigational information are packaged into an OPS Publication.



Note: Oxygen XML Editor plugin supports both OPF 2.0 and OPF 3.0.

Chapter 8

Author Developer Guide

Topics:

- [Simple Customization Tutorial](#)
- [Advanced Customization Tutorial - Document Type Associations](#)
- [CSS Support in Author](#)
- [Example Files Listings - The Simple Documentation Framework Files](#)
- [Author Component](#)
- [Creating and Running Automated Tests](#)

The Author editor of Oxygen XML Editor plugin was designed to bridge the gap between the XML source editing and a friendly user interface. The main achievement is the fact that the Author combines the power of source editing with the intuitive interface of a text editor.

This guide is targeted at advanced authors who want to customize the Author editing environment and is included both as a chapter in the Oxygen XML Editor plugin user manual and as a separate document in [the Author SDK](#).

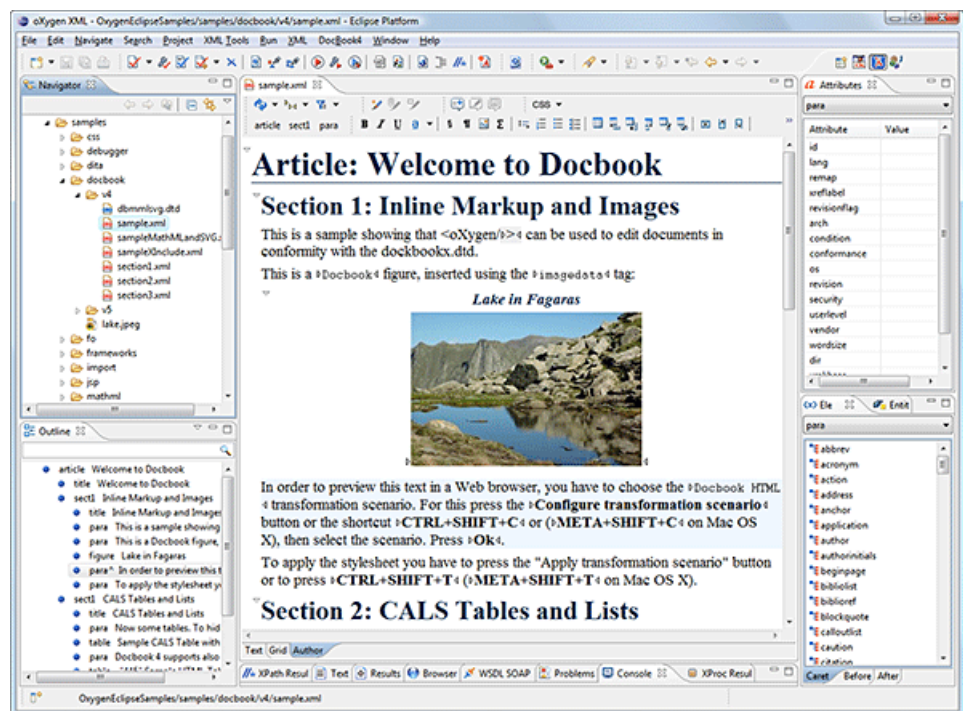


Figure 230: Oxygen XML Editor plugin Author Visual Editor

Although Oxygen XML Editor plugin comes with already configured frameworks for DocBook, DITA, TEI, XHTML, you might need to create a customization of the editor to handle other types of documents. The 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 in 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 refer the CSS through an `xml-stylesheet` processing instruction.

2. Fully configure a document type association. This involves putting together the CSSs, the XML schemes, actions, menus, etc, bundling them and distributing an archive. The CSS and the GUI elements are settings of the Oxygen XML Editor plugin Author. The other settings like the templates, catalogs, transformation scenarios are general settings and are enabled whenever the association is active, no matter the editing mode (Text, Grid or Author).

Both approaches will be discussed in the following sections.

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"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="report">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="description"/>
        <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 into the Author. This is done using Cascading Style Sheets or CSS on short. 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. But any block that contains inline boxes is arranging its children in a horizontal flow. That is why the paragraph lines are also blocks, but the traditional "bold" and "italic" sections are represented as inline 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 "table-row" style. Similarly, the "row" elements must contain elements with "table-cell" style.

To make it easy to understand, the following section describes the way each element from the above schema is formatted using a CSS file. Please note that this is just one from an infinite number of possibilities of formatting the content.

report This element is the root element of the 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 larger font. The **block** display should also be used - the next elements will be placed below it, and change its font 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 will have the same **block** display. To make it stand out the background color is changed.

```
description {
  display:block;
  background-color:#EEEEFF;
  color:black;
}
```

line A line of text in the description. A specific aspect is not defined for it, just indicate that the display should be **block**.

```
line {
  display:block;
}
```

important The `important` element defines important text from the description. Because it can be mixed with text, its display property must be set to **inline**. To make it easier to spot, the text will be emphasized.

```
important {
  display:inline;
  font-weight:bold;
}
```

results The `results` element shows the list of `test_names` and the result 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 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;
}
```

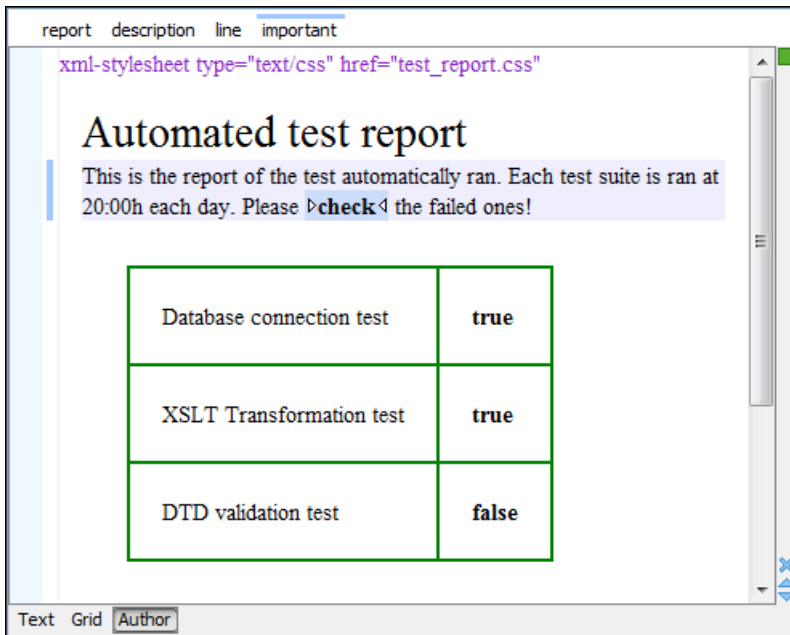


Figure 231: A report opened in the Author



Note:

You can edit attributes in-place in the Author mode using *form-based controls*.

The XML Instance Template

Based on the XML Schema and the CSS file the Author can help the content author in loading, editing and validating the test reports. An XML file template must be created, 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 refer 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 `xml-stylesheet` 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 plugin is highly customizable. Practically you can associate an entire class of documents (grouped logically by some common features like namespace, root element name or filename) to a bundle consisting of CSS stylesheets, validation schemas, catalog files, new files templates, transformation scenarios and even custom actions. The bundle is called *document type* and the association is called *Document Type Association* or, more generically, *framework*.

In this tutorial, we create a **Document Type Association** 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 Author SDK zip](#), available for download at http://www.oxygenxml.com/oxygen_sdk.html.



Note: The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

Document Type

A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the **Author** mode for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both **Author** mode and Text mode;
- CSS stylesheet(s) for rendering XML documents in **Author** mode;
- user actions invoked from toolbar or menu in **Author** mode;
- predefined scenarios used for transformation of 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.

The tagless editor comes with some predefined document types already configured when the application is installed on the computer. These document types describe well-known XML frameworks largely used today for authoring XML documents. Editing a document which conforms to one of these types is as easy as opening it or creating it from one of the predefined document templates which also come with Oxygen XML Editor plugin.

To see our video demonstration about configuring a framework in Oxygen XML Editor plugin, go to <http://oxygenxml.com/demo/FrameworkConfiguration.html>.

Document Type Settings

You can add a new *Document Type Association* or edit the properties of an existing one from the **Options > Preferences > Document Type Association** option pane. All the changes can be made into the *Document type* edit dialog.

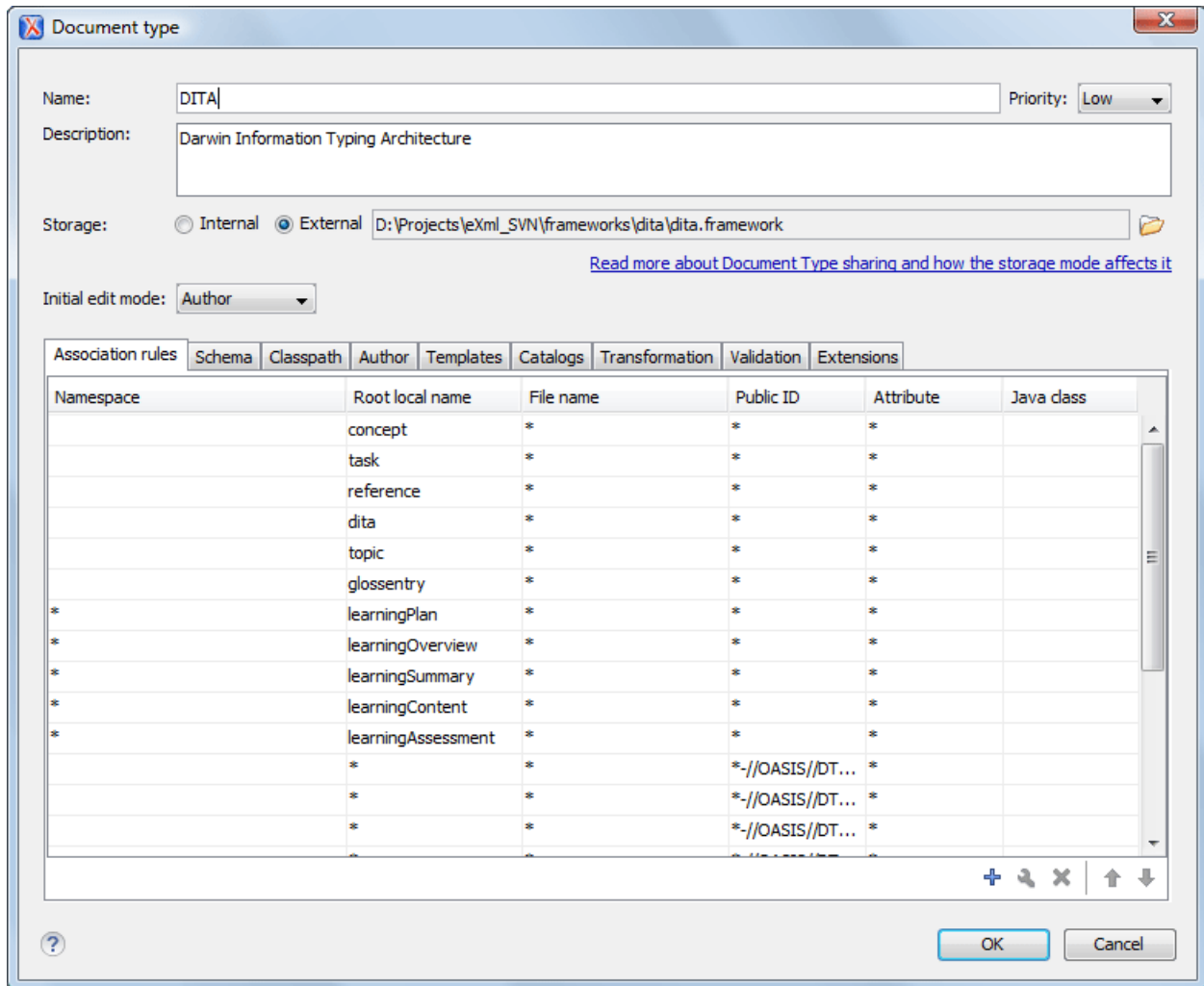


Figure 232: The Document Type

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: 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: The frameworks having the same priority are alphabetically sorted.

- **Description** - The document type description displayed as a tooltip in the *Document Type Association table*.
- **Storage** - The location where the document type is saved. If you select the **External** storage, the document type is saved in the specified file with a mandatory `framework` extension, located in a subfolder 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 plugin project file (for Project-level Document Type Association Options) or in the Oxygen XML Editor plugin internal options (for Global-level Document Type Association Options). You can change the Document Type Association Options level in the *Document Type Association panel*.
- **Initial edit mode** - Allows you to select the initial editing mode (**Editor specific**, **Text**, **Author**, **Grid** and **Design** (available only for the W3C XML Schema editor)) for this document type. If the **Editor specific** option is selected, the initial edit mode is determined depending on the editor type. You can find the mapping between editors and edit modes in the *Edit modes preferences page*. You can decide to impose an initial mode for opening files which 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. To open this page, go to **Window > Preferences > oXygen XML Editor > 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 Java class that is called for finding 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.

Configuring Actions, Menus and Toolbars

You can change the Author toolbars and menus to gain a productive editing experience. You can create a set of actions that are specific to a document type, using the **Document Type** dialog.

In the example with the `sdf` framework, you created the stylesheet and the validation schema. Now let's add some actions to insert a `section` and a `table`. To add a new action, follow the procedure:

1. Go to **Options > Preferences > Document Types Association** and click the framework for which you want to create an action.
2. Click **Edit** and in the **Document Type** dialog go to the **Author** tab, then go to **Actions**.
3. Click the **+** **New** and use the *Action dialog* to create an action.

Creating the Insert Section Action

This section presents all the steps that you need to follow, to define the **Insert Section** action. We assume the icon files `§ (Section16.png)` for the menu item and `§ (Section20.png)` for the toolbar, are already available. Although you could use the same icon size for both 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.

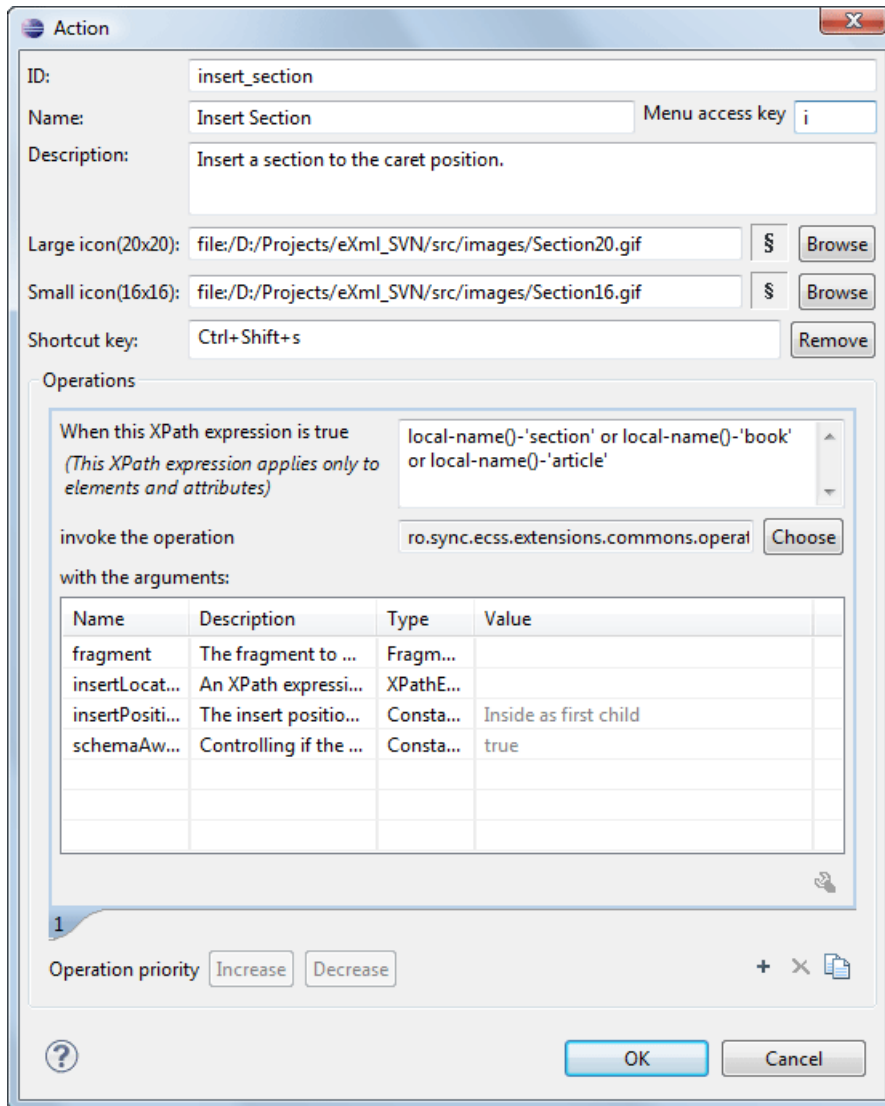


Figure 233: The Action Dialog

1. Set the **ID** field to **insert_section**. This is a 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) 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 `${frameworks} / sdf / Section20.png`. A good practice is to store the image files inside the framework directory and use *editor variable* `${frameworks}` 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 refer the images not by the file name, but by their relative path location in the class-path.

If the image file `Section20.png` is located in the **images** directory inside the jar archive, you can refer to it by using `/images/Section20.png`. The jar file must be added into the **Classpath** list.

6. Set the **Small icon (16x16)** field to `${frameworks} / sdf / Section16.png`.

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"**.

The Insert Table Action

You will create an action that inserts into the document a table with three rows and three columns. The first row is the table header. Similarly to the insert section action, you will use the `InsertFragmentOperation`.

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 `${frameworks} / sdf / toolbarIcon.png`.
6. Set **Menu icon** to `${frameworks} / sdf / menuIcon.png`.
7. Set **Shortcut key** to `Ctrl (Meta on Mac OS)+Shift+T`.
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()]
```

Configuring the Toolbars

Now that you have defined the *Insert Section* action and the *Insert Table* action, you can add them to the toolbar. You can configure additional toolbars on which to add your custom actions.

1. Open the Document Type edit dialog for the **SDF** framework and select on the **Author** tab. Next click on the **Toolbar** label.

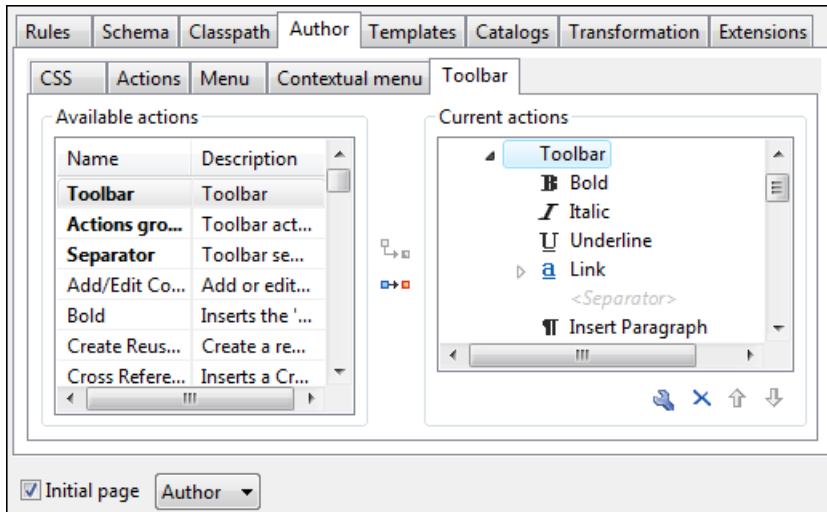
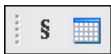


Figure 234: Configuring the Toolbar

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 235: 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.

Configuring the Main Menu

Defined actions can be grouped into customized menus in the Oxygen XML Editor plugin menu bar.

1. Open the Document Type dialog 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.

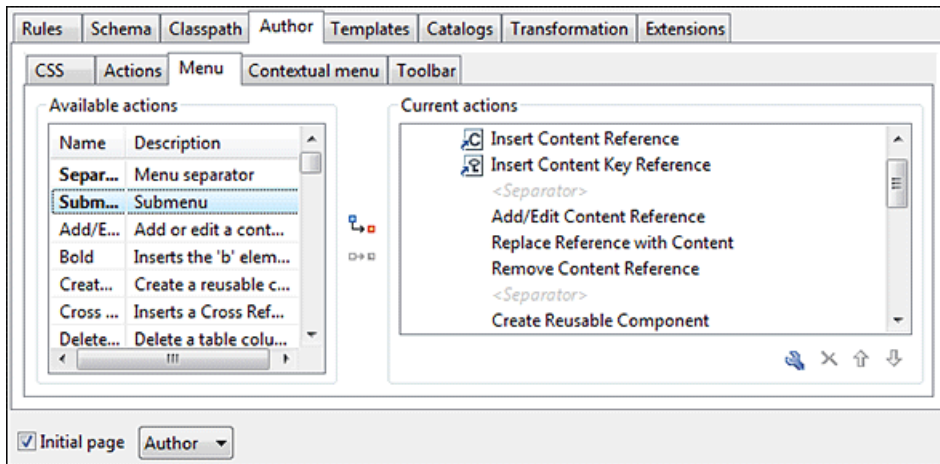


Figure 236: Configuring the Menu

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).

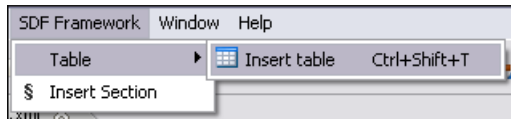


Figure 237: Author Menu

Configuring the Contextual Menu

The contextual menu is shown when you right click (**Ctrl (Meta on Mac OS) + mouse click** on Mac) in the Author editing area. In fact you are configuring the bottom part of the menu, since the top part is reserved for a list of generic actions like Copy, Paste, Undo, etc.

1. Open the Document Type dialog for the **SDF** framework and click on the **Author** tab. Next click on the **Contextual Menu** label.
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.

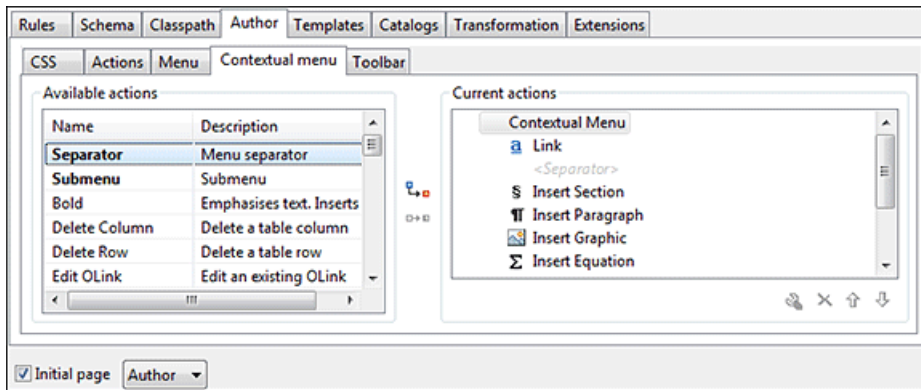


Figure 238: Configuring the Contextual Menu

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.

Customize Content Completion

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** edit dialog for the **SDF** framework and select the **Author** tab. Next click on the **Content Completion** tab.

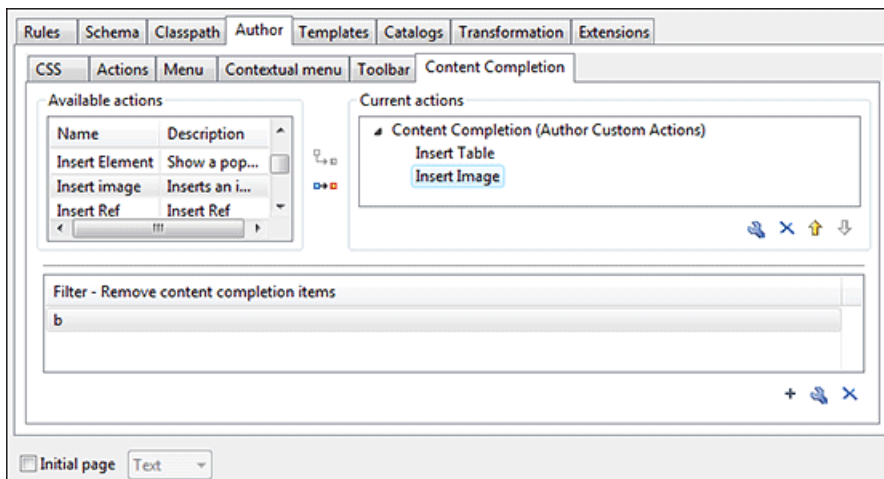


Figure 239: Customize Content Completion

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. The following dialog appears, giving you the possibility to select where to provide the selected action:

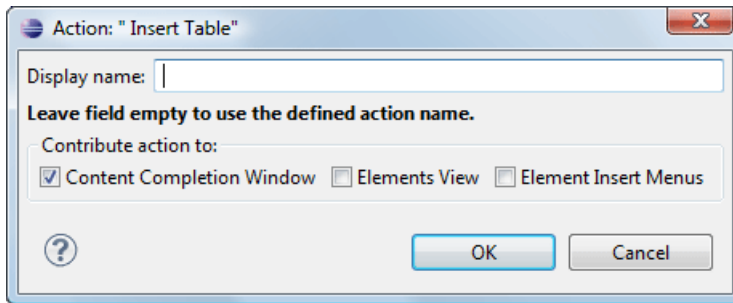


Figure 240: Insert action dialog

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 following dialog is displayed, allowing you to input the item name and to choose the controls that filter it.

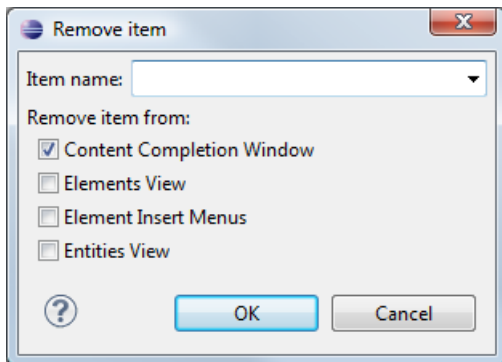


Figure 241: Remove item dialog

Author Default Operations

Below are listed all the operations and their arguments:

- **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 411.

- **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 412.

- **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.

- **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 element with specific attributes. It is useful to implement toggle actions like highlighting text as bold, italic, or underline. 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 if the insertion is schema aware or not.

- **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.

- **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.

- **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.

Author operations can take parameters that might contain the following editor variables:

- **`\${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. Can have one of the following values:
 - url - input is considered an URL. Oxygen XML Editor plugin checks that the URL is valid before passing it to the transformation;
 - password - input characters are hidden;
 - generic - the input is treated as generic text that requires no special handling;
 - relative_url - input is considered an URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing;



Note: You can use the `$ask` editor variable in file templates. In this case, Oxygen XML Editor plugin keeps an absolute URL.

- combobox - displays a dialog that contains a non-editable combo-box;
- editable_combobox - displays a dialog that contains an editable combo-box;
- radio - displays a dialog that contains radio buttons;
- 'default-value' - optional parameter. Provides a default value in the input text box;

Examples:

- ``${ask('message')}`` - Only the message displayed for the user is specified.
 - ``${ask('message', generic, 'default')}`` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
 - ``${ask('message', password)}`` - 'message' is displayed, the characters typed are masked with a circle symbol.
 - ``${ask('message', password, 'default')}`` - same as before, the default value is 'default'.
 - ``${ask('message', url)}`` - 'message' is displayed, the parameter type is URL.
 - ``${ask('message', url, 'default')}`` - same as before, the default value is 'default'.
- **`\${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; An 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;
 - **`\${cfd}`** - 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 `frameworks` subfolder of the Oxygen XML Editor plugin installation folder;
 - **`\${pd}`** - Current project folder as file path. Usually the current folder selected in the Project View;
 - **`\${oxygenInstallDir}`** - Oxygen XML Editor plugin 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 plugin 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):

Result:

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:

Gives the result document:

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.

How to Add a Custom Operation to an Existing Document Type

This task explains how to add a custom Author operation to an existing document type.

1. Download the Author SDK toolkit:http://www.oxygenxml.com/developer.html#XML_Editor_Authoring_SDK
2. Create a Java project with a custom implementation of `ro.sync.ecss.extensions.api.AuthorOperation` which performs your custom operation and updates the **Author** mode using our API like:
`AuthorAccess.getDocumentController().insertXMLFragment.`
3. Pack the operation class inside a Java *jar* library.
4. Copy the *jar* library to the `OXYGEN_INSTALL_DIR/frameworks/framework_dir` directory.
5. Go to Oxygen **Preferences** > **Document Type Association** page and edit the document type (you need write access to the `OXYGEN_INSTALLATION_DIR`).
 - a) In the **Classpath** tab, add a new entry like: `${frameworks}/docbook/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_INSTALL_DIR/frameworks/framework_dir` directory.

Java API - Extending Author Functionality through Java

Oxygen XML Editor plugin 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 nodeset.

The following sections contain the Java programming interface (API) available to the developers. You will need the *Oxygen Author SDK* available *on the Oxygen XML Editor plugin 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 plugin 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 plugin 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 plugin. Otherwise the classes may not be loaded by the Java virtual machine. For example if you run 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. Step by Step Example. Simple Use of a Dialog 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 with a text field, in which the user can enter a full URL, or he can browse for a local file.

1. Create a new Java project, in your IDE of choice. Create the `lib` folder in the project folder. Copy the `oxygen.jar` file from the `{oxygen_installation_directory}/lib` folder into the newly created `lib` folder. `oxygen.jar` contains the Java interfaces you have to implement and the API needed to access the Author features.
2. Create 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 plugin 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 plugin 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.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.getCaretOffset();
authorAccess.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 Author SDK zip](#) available for download [on the Oxygen XML Editor plugin 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>
```

- Copy the sdf.jar file into the frameworks / sdf folder.
- Add the sdf.jar to the Author class path. To do this, open the **Options > Preferences > Document Type Association** dialog, select **SDF** and press the **Edit** button.
- Select the **Classpath** tab in the lower part of the dialog and press the **+ Add** button. In the displayed dialog enter the location of the jar file, relative to the Oxygen XML Editor plugin frameworks folder.
- 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.
- 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 **\${frameworks}/sdf/toolbarImage.png**.
 - Set **Menu icon** to **\${frameworks}/sdf/menuImage.png**.
 - Set **Shortcut key** to **Ctrl (Meta on Mac OS)+Shift+i**.

- 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**.

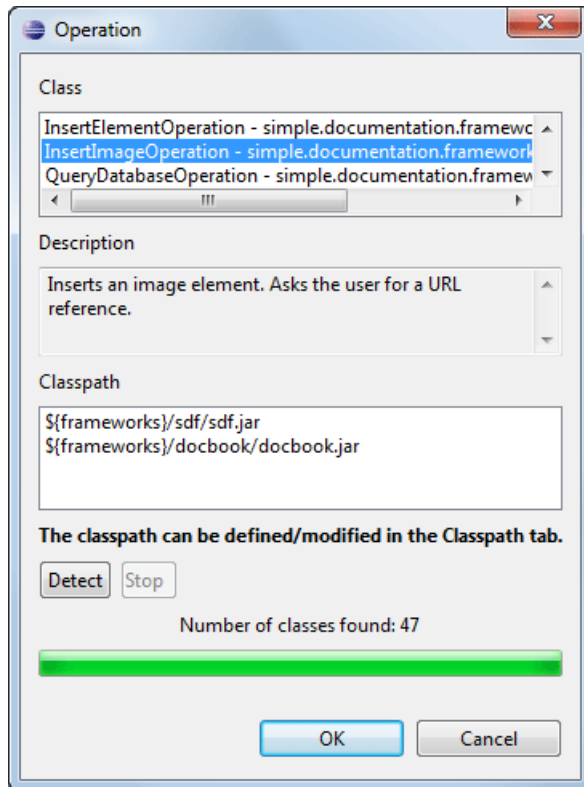


Figure 242: Selecting the Operation

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 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. Create a new Java project in your preferred IDE. Create the `lib` folder in the Java project directory and copy the `oxygen.jar` file from the `{oxygen_installation_directory}/lib` directory.
2. Create the class `simple.documentation.framework.QueryDatabaseOperation`. This class must implement 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.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 Author SDK zip](#) available for download [on the Oxygen XML Editor plugin 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. For this, Open the options Document Type Dialog, select **SDF** and press the **Edit** button. Select the **Classpath** tab in the lower part of the dialog.
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 `$(frameworks)/sdf/TableDB20.png` (image  `TableDB20.png` is already stored in the `frameworks / sdf` folder).
 - Leave empty the **Menu icon**.
 - Set **shortcut key** to **Ctrl (Meta on Mac OS)+Shift+C**.
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.

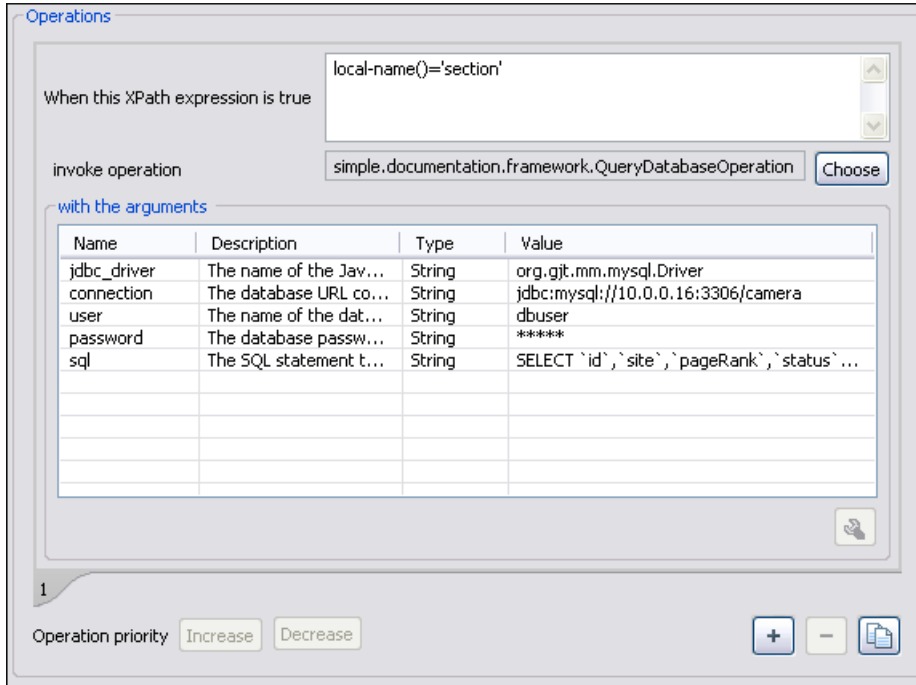


Figure 243: Java Operation Arguments Setup

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.

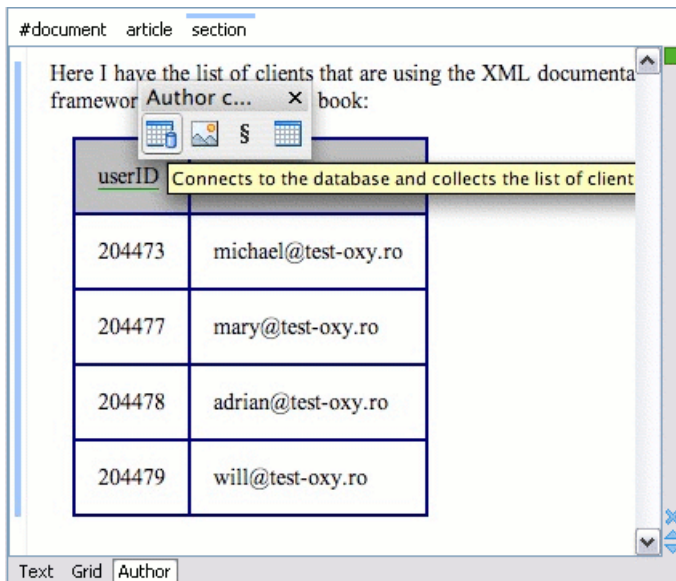


Figure 244: Table Content Extracted from the Database

Editing attributes in-place using form controls

To edit attributes in the **Author** mode, use the [Attributes View](#) or the in-place attributes editing dialog.

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 editing attributes in-place using some of these default implementations.

Localizing Frameworks

Oxygen XML Editor plugin 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 plugin (in **Options > Preferences > Global > Language** 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 plugin matches the GUI language with the language set in the `translation.xml` file. In case 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:

- framework actions;
- menu entries;
- contextual menus;
- toolbar;
- static CSS content.

For example, if you want to localize the bold action go to **Options > Preferences > Document Type Association**.

Open the **Document type** dialog, 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 for a certain key the associated value. In this way all the dialogs that you present from your custom operations can have labels translated in different languages.

You can also refer 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_INSTALL_DIR]/frameworks/docbook/i18n/translation.xml`. In the **Classpath** list corresponding to the Docbook document type the following entry was added: `${framework}/i18n/`.

In **Options > Preferences > Document Type Association > Author > Actions**, you can see how the DocBook actions are defined to use these keys for their name and description. If you look in the Java class `ro.sync.ecss.extensions.docbook.table.SADocbookTableCustomizerDialog` available in the Author SDK, you can see how the new `ro.sync.ecss.extensions.api.AuthorResourceBundle` API is used to retrieve localized descriptions for different keys.

How to Deploy a Plugin or a Framework as an Oxygen XML Editor plugin Add-on

To deploy a plugin or a framework as an Oxygen XML Editor plugin add-on:

1. Pack it as a ZIP file or a *Java Archive (JAR)*. Please note that you should pack the entire root directory not just it's 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_install_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).

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 plugin provides. To use this template, go to **File > New** and select the **Oxygen add-ons update site** template.
4. Copy the ZIP 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 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 <code>\${frameworks}/sdf/schema/sdf.xsd</code> . We should use the <code>\${frameworks}</code> editor variable in the schema URI path instead of a full path in order to be valid for different Oxygen XML Editor plugin 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 plugin 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;
}
```

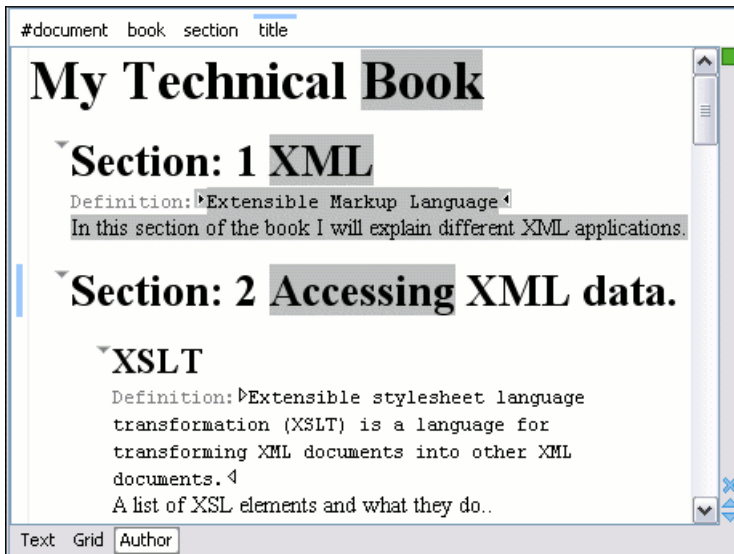


Figure 245: A sample of nested sections and their titles.

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 plugin 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 plugin.

```
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 plugin 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 plugin Author handles both absolute and relative specified URLs. If the image has an *absolute* URL location (e.g: "`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 (e.g: "`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);
}
```

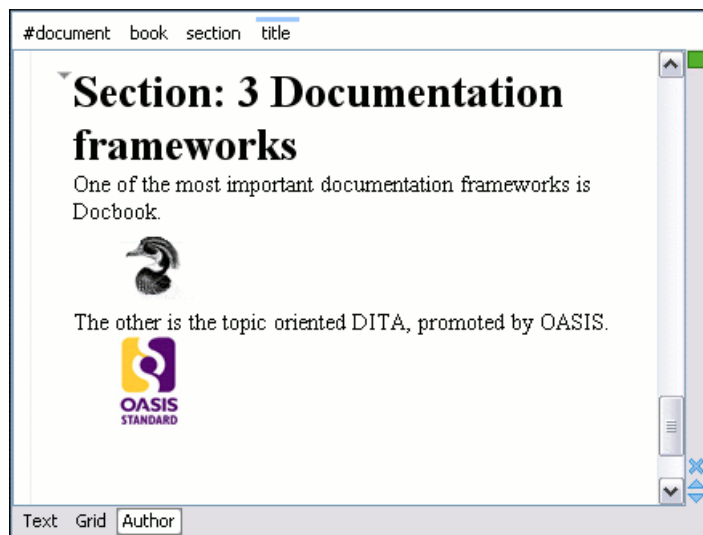


Figure 246: Samples of images in Author

Testing the Document Type Association

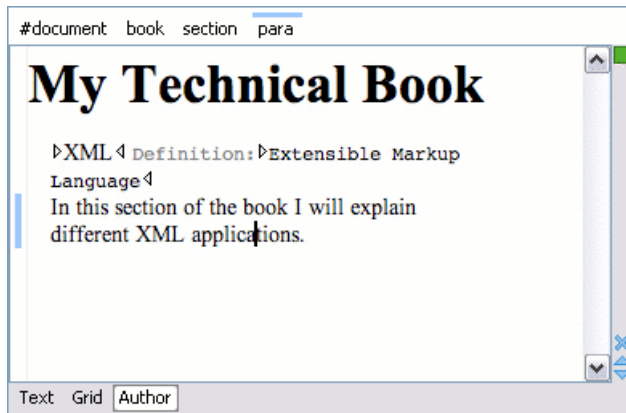
To test the new Document Type create an XML instance that is conforming with the *Simple Documentation Framework* association rules. You will not specify an XML Schema location directly in the document, using an `xsi:schemaLocation` attribute; Oxygen XML Editor plugin will detect instead its associated 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 plugin 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_installation_directory}/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 framework customizations are located. Each subdirectory contains files related to a specific type of XML documents: schemas, catalogs, stylesheets, CSSs, 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 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 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 plugin `frameworks` directory. Select the `sdf` directory and make an archive from it. Move it to another Oxygen XML Editor plugin installation (eventually on another computer). Extract it in the `frameworks` directory. Start Oxygen XML Editor plugin 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 plugin 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 Document Type Association dialog shall always be stored inside the `sdf` directory). If your external storage points somewhere else Oxygen XML Editor plugin 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-install-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>
      <thead>
        <tr>
          <th></th>
          <th></th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Company</td>
          <td>Date</td>
        </tr>
      </tbody>
    </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>
```

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 `#{cfd}`, `#{cf}`, `#{cfu}`, and `#{cfdu}` editor variables when you save your documents in a data base.

3. Open the Document Type dialog 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** dialog. 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, like 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, the command line of an external tool) to make a command or a parameter generic and reusable 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 plugin commands of external engines or other external tools, in transformation scenarios, validation scenarios, and Author operations:

- **`\${oxygenHome}`** - Oxygen XML Editor plugin installation folder as URL;
- **`\${oxygenInstallDir}`** - Oxygen XML Editor plugin installation folder as file path;
- **`\${frameworks}`** - The path (as URL) of the `frameworks` subfolder of the Oxygen XML Editor plugin install folder;
- **`\${frameworksDir}`** - The path (as file path) of the `frameworks` subfolder of the Oxygen XML Editor plugin installation folder;
- **`\${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;
- **`\${cfd}`** - 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;
- **`\${cfu}`** - The path of the current file as a URL. The current file is the one currently opened and selected;
- **`\${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, Mac 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; An 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. Can have one of the following values:
 - url - input is considered an URL. Oxygen XML Editor plugin checks that the URL is valid before passing it to the transformation;
 - password - input characters are hidden;
 - generic - the input is treated as generic text that requires no special handling;
 - relative_url - input is considered an URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing;



Note: You can use the `$ask` editor variable in file templates. In this case, Oxygen XML Editor plugin keeps an absolute URL.

- combobox - displays a dialog that contains a non-editable combo-box;
- editable_combobox - displays a dialog that contains an editable combo-box;
- radio - displays a dialog that contains radio buttons;
- 'default-value' - optional parameter. Provides a default value in the input text box;

Examples:

- `${ask('message')}` - Only the message displayed for the user is specified.
 - `${ask('message', generic, 'default')}` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
 - `${ask('message', password)}` - 'message' is displayed, the characters typed are masked with a circle symbol.
 - `${ask('message', password, 'default')}` - same as before, the default value is 'default'.
 - `${ask('message', url)}` - 'message' is displayed, the parameter type is URL.
 - `${ask('message', url, 'default')}` - same as before, the default value is 'default'.
- **`${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.

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 a , 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 [Preferences page](#).

Create Your Own Stylesheet Templates

Oxygen XML Editor plugin 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 plugin installation directory.
- Open Oxygen XML Editor plugin 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 `xmlns:schemaLocation` attribute, but instead you let the editor use the schema from the association. However there are cases in which you must refer for instance 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_installation_directory} / 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.

To test the catalog settings, restart Oxygen XML Editor plugin 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:

```
<xs: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:

```
xmlns: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 would help the content authors publish their work in different formats. Being contained in the **Document Type Association** the scenarios can be distributed along with the actions, menus, toolbars, catalogs, etc.

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 **Options/Preferences/Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework then choose the **Transformation** tab. Click the **New** button.

In the **Edit Scenario** dialog, fill 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 `${frameworks}/sdf/xsl/sdf.xsl`.
- Set the **Transformer** to *Saxon 9B*.

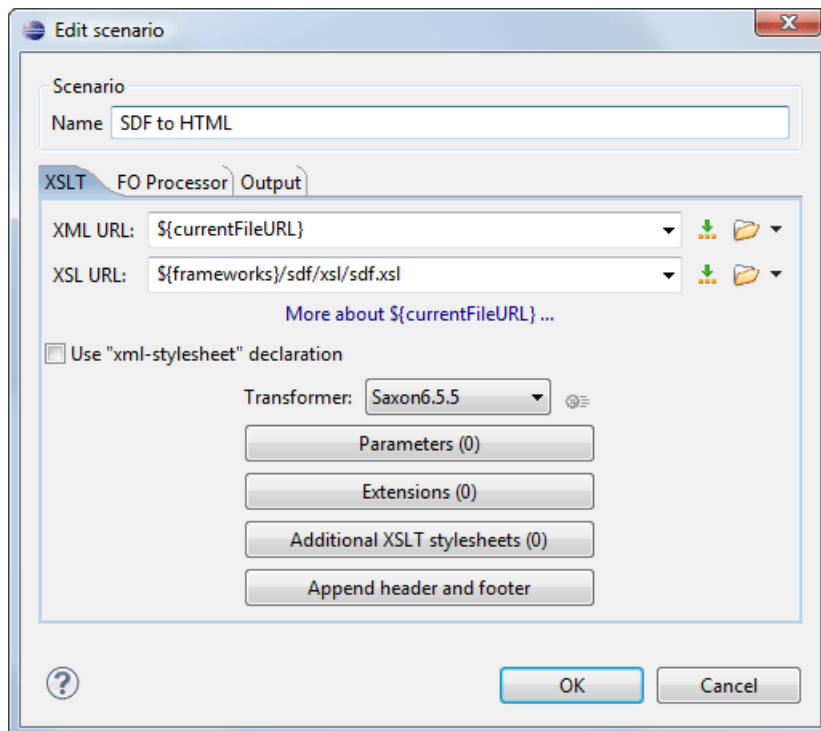


Figure 247: Configuring a transformation scenario

4. Change to the **Output** tab. Configure the fields as follows:

- Set the **Save as** field to $\${cfd}/\${cfn}.html$. 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: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes precedence over the default system application settings.

- Enable the **Saved file** option.

Now the scenario is listed in the **Transformation** tab:

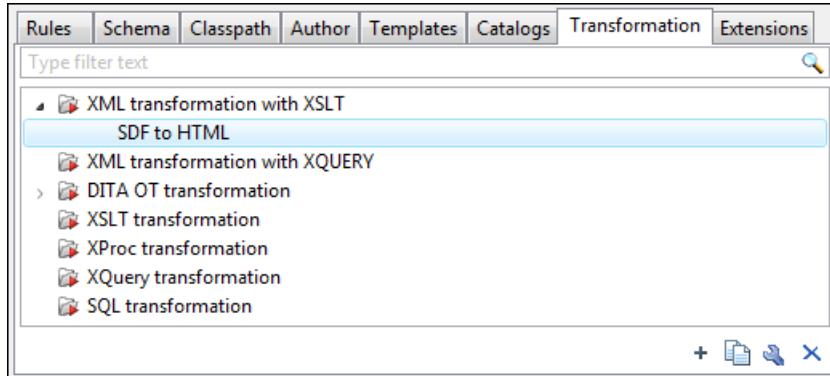


Figure 248: The transformation tab

To test the transformation scenario you just created, open the **SDF XML** sample from the *Example Files Listings*. Click the **Apply Transformation Scenario(s)** button to display the **Configure Transformation Scenario(s)** dialog. Its scenario list contains the scenario you defined earlier *SDF to HTML*. Click it then choose **Transform now**. The HTML file should be saved in the same folder as the XML file and displayed in the browser.

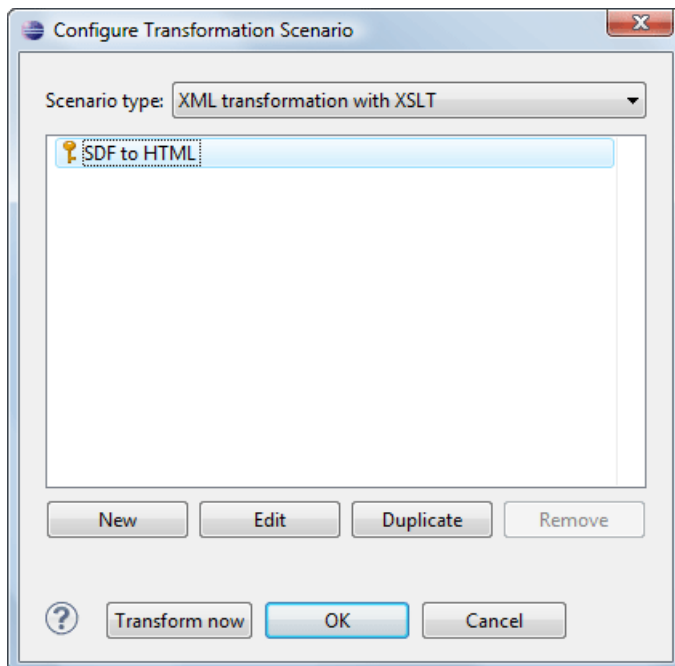




Figure 249: Selecting the predefined scenario

Configuring Validation Scenarios

You can distribute a framework with a series of already configured validation scenarios. Also, this provides enhanced validation support allowing 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 **Options/Preferences/Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework, then choose the **Validation** tab. This tab holds a list of document types for which you can define validation scenarios. To set one of the validation scenarios as default for a specific document type, select it and press  /  **Toggle default**.
2. Press the **New** button to add a new scenario.
3. Press the **Add** button to add a new validation unit with default settings. The dialog that lists all validation units of the scenario is opened.

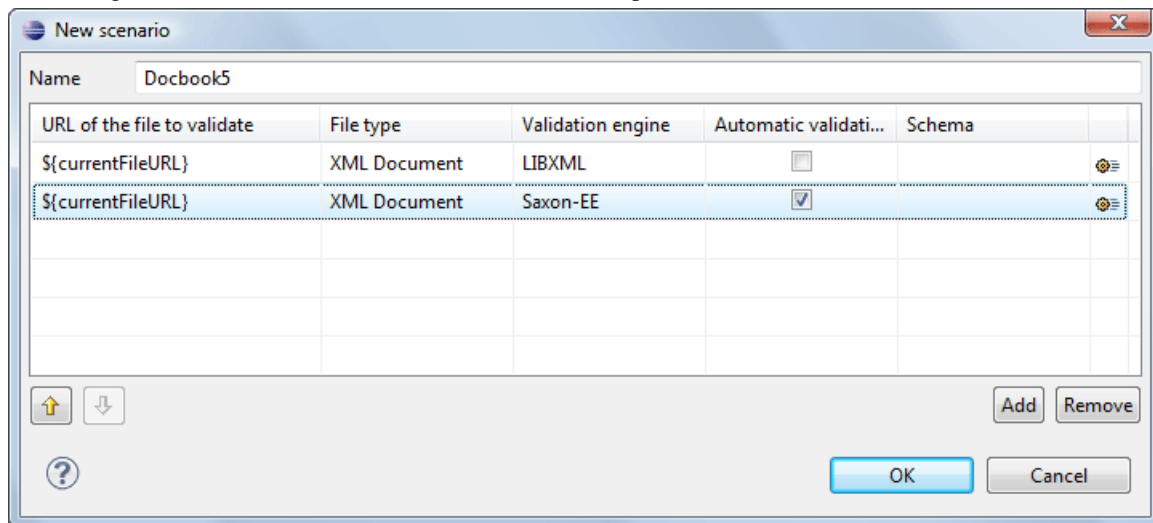


Figure 250: Add / Edit a Validation Unit

The table holds the following information:

- **Storage** - allows you to create a scenario at project level, or as global;
 - **URL of the file to validate** - the URL of the main module which 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 validated in the current validation unit. Oxygen XML Editor plugin 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 plugin for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the default engine executes the validation. This engine is set in **Preferences** pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, and others) instead of a validation scenario;
 - **Automatic validation** - if this option is checked, then the validation operation defined by this row of the table is applied also by *the automatic validation feature*. If the **Automatic validation** feature is *disabled in Preferences* then this option does not take effect as the Preference setting has higher priority;
 - **Schema** - the this option becomes active when you set the **File type** to **XML Document**;
 - **Settings** - opens the **Specify Schema** dialog box, allowing 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. Edit the URL of the main validation module.
Specify the URL of the main module:

- browsing for a local, remote, or archived file;
- using an *editor variable* or a *custom editor variable*, available in the following pop-up menu, opened after pressing the  button:

```

${start-dir} - Start directory of custom validator
${standard-params} - List of standard parameters
${cfn} - The current file name without extension
${currentFileURL} - The path of the currently edited file (URL)
${cfdu} - The path of current file directory (URL)
${frameworks} - Oxygen frameworks directory (URL)
${pdu} - Project directory (URL)
${oxygenHome} - Oxygen installation directory (URL)
${home} - The path to user home directory (URL)
${pn} - Project name
${env(VAR_NAME)} - Value of environment variable VAR_NAME
${system(var.name)} - Value of system variable var.name

```

Figure 251: Insert an Editor Variable

5. Select the type of the validated document.
Note that it 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 *automatic validation feature is turned on in Preferences*.
8. Choose what schema is used during validation: the one 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.

Configuring an Extensions Bundle

Starting with Oxygen XML Editor plugin 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 go to **Options > Preferences > 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. Therefore 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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_installation_directory}/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, 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 the [Implementing an Author Schema Aware Editing Handler](#) section 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 SDFSSchemaManagerFilter();
}

```

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 referred 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 plugin Eclipse plugin and standalone application. The Text mode corresponding listener is available only for Oxygen XML Editor plugin 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.

- 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 referred resource is `location`. To be able to obtain the content of the referred 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.

- 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.

- 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`.

- 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 plugin 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 clicking on the **Extensions** tab. Press the **Choose** button and select from the displayed dialog the name of the class: `SDFExtensionsBundle`.



Note: The complete source code can be found in the Simple Documentation Framework project, included in the [Oxygen Author SDK zip](#) available for download [on the Oxygen XML Editor plugin 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.

Preserve Style and Format on Copy and Paste from External Applications

Styled content can be inserted in the Author editor by copying or dragging it from:

- Office-type applications (**Microsoft Word** and **Microsoft Excel**, **OpenOffice.org Writer** and **OpenOffice.org Calc**);
- web browsers (like **Mozilla Firefox** or **Microsoft Internet Explorer**);
- the **Data Source Explorer** view (where resources are available from WebDAV or CMS servers).

The styles and general layout of the copied content like: sections with headings, tables, list items, bold, and italic text, hyperlinks, are preserved by the paste operation by transforming them to the equivalent XML markup of the target document type. This is available by default in the following *predefined document types*: [DITA](#), [DocBook 4](#), [DocBook 5](#), [TEI 4](#), [TEI 5](#), [XHTML](#).

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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

```
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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

```
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().isEnabledSmartTyping()) {
        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 Author SDK zip](#) available for download [on the Oxygen XML Editor plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

```
import java.util.List;

import ro.sync.contentcompletion.xml.CIAttribute;
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 SDFSchemasManagerFilter implements SchemaManagerFilter {
```

You can implement the various callbacks of the interface either by returning the default values given by Oxygen XML Editor plugin 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.CIAttribute` for the element provided by the current `ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext` context. For example, the `SDFSchemasManagerFilter` 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<CIAttribute> filterAttributes(List<CIAttribute> 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())) {
            CIAttribute frameAttribute = new CIAttribute();
            frameAttribute.setName("frame");
            frameAttribute.setRequired(false);
            frameAttribute.setFixed(false);
            frameAttribute.setDefaultValue("void");
            if (attributes == null) {
                attributes = new ArrayList<CIAttribute>();
            }
            attributes.add(frameAttribute);
        }
    }
    return attributes;
}
```

The elements that can be inserted in a specific context can be filtered using the `filterElements` method. The `SDFSchemasManagerFilter` 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 Author SDK zip* available for download [on the Oxygen XML Editor plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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 IDElementLocator(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 <i>NCName</i> 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(elemID/2/3/4).");
        }
        i++;
    }

    if(Character.isDigit(xpointerPath[0].charAt(0))){
        // This is the case when xpointer have the following pattern /1/5/7
        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

<i>uri</i>	The namespace URI, or the empty string if the element has no namespace URI or if namespace processing is disabled.
<i>localName</i>	Local name of the element.
<i>qName</i>	Qualified name of the element.
<i>atts</i>	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[] atts) {
    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 (startWithElementID) {
    // This the case when xpointer path starts with an element ID.
    String xpointerElement = xpointerPath[0];
    for (int i = 0; i < atts.length; i++) {
        if(xpointerElement.equals(atts[i].getValue())){
            if(idVerifier.hasIDType(
                localName, uri, atts[i].getQName(), atts[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[] atts) {
    boolean elementFound = false;
    for (int i = 0; i < atts.length; i++) {
        if (link.equals(atts[i].getValue())) {
            if ("xml:id".equals(atts[i].getQName())) {
                // xml:id attribute
                elementFound = true;
            } else {
                // check if attribute has ID type
                String attrLocalName =
                    ExtensionUtil.getLocalName(atts[i].getQName());
                String attrUri = atts[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 Default Frameworks project, included in the *Oxygen Author SDK zip* available for download [on the Oxygen XML Editor plugin website](#).

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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

Table 7: Interfaces for the DnD 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 refer. In our case the element which has references is **ref** and the attribute indicating the referred resource is **location**. You will have to implement a Java extension class for obtaining the referred resources.

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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 {

```

- 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;
}
```

- The method `getDisplayname` returns the display name of the node that contains the expanded referred content. It takes as argument an `AuthorNode` that represents the node for which the display name is needed. The referred 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.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;
}
```

5. 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();
            }
        }
    }
    return id;
}
```

6. The method `getReferenceSystemID` should return the `systemID` of the referred 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.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 Author SDK zip](#) available for download [on the Oxygen XML Editor plugin website](#).

In the listing below, the XML document contains the `ref` element:

```
<ref location="referred.xml">Reference</ref>
```

When no reference resolver is specified, the reference has the following layout:

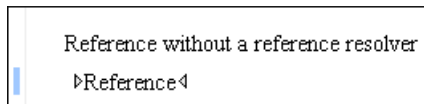


Figure 252: Reference with no specified reference resolver

When the above implementation is configured, the reference has the expected layout:

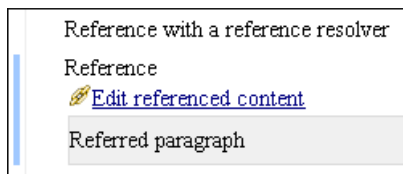


Figure 253: Reference with reference resolver

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 plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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 plugin 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 plugin 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 plugin 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 453.

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 447, if you use the table element attribute **width** Oxygen XML Editor plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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;
}
```

6. 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 Author SDK zip* available for download [on the Oxygen XML Editor plugin website](#).

In the listing below, the XML document contains the table element:

```

<table width="300">
  <customcol width="50.0px"/>
  <customcol width="1*"/>
  <customcol width="2*"/>
  <customcol width="20%"/>
  <header>
    <td>C1</td>
    <td>C2</td>
    <td>C3</td>
    <td>C4</td>
  </header>
  <tr>
    <td>cs=1, rs=1</td>
    <td>cs=1, rs=1</td>
    <td rowspan="2">cs=1, rs=2</td>
    <td rowspan="3">cs=1, rs=3</td>
  </tr>
  <tr>
    <td>cs=1, rs=1</td>
    <td>cs=1, rs=1</td>
  </tr>
  <tr>
    <td colspan="3">cs=3, rs=1</td>
  </tr>
</table>

```

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 254: 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 255: 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 447, you need to indicate Oxygen XML Editor plugin a method to determine the cell spanning. If you use the cell element attributes **rowspan** and **colspan** or **rows** and **cols**, Oxygen XML Editor plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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) {
}
```

3. 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;
}
```

4. 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;
}
```

5. 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 Author SDK zip* available for download [on the Oxygen XML Editor plugin website](#).

6. 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:

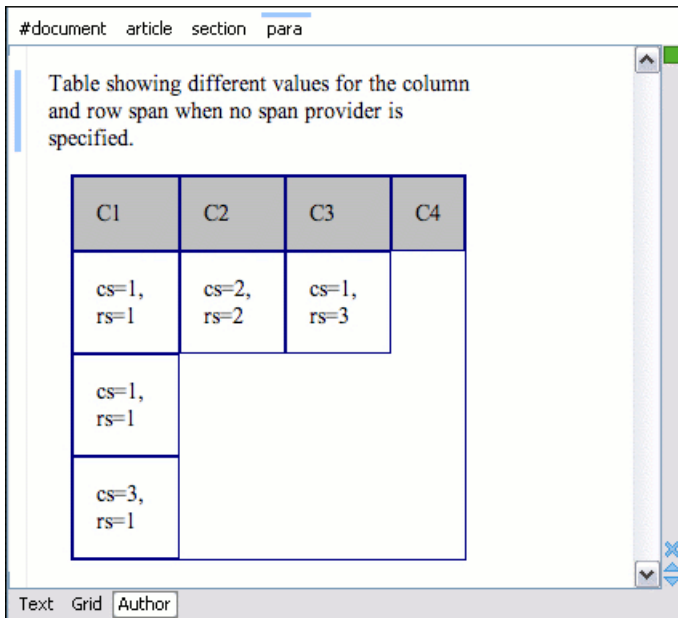


Figure 256: Table layout when no cell span provider is specified

When the above implementation is configured, the table has the correct layout:

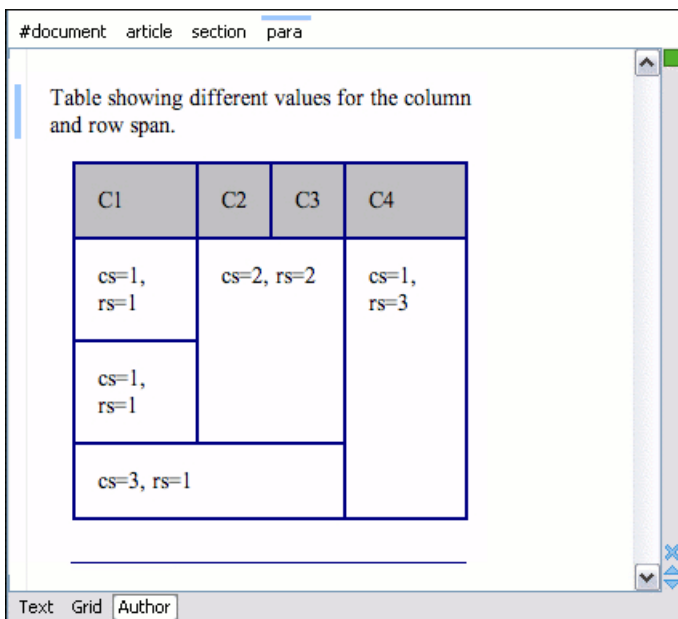


Figure 257: Cells spanning multiple rows and columns.

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 447 section which describes the CSS properties needed for defining a table, you need to indicate Oxygen XML Editor plugin 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 plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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 Author SDK zip* available for download [on the Oxygen XML Editor plugin 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 258: 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:

 **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

- **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 plugin 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 plugin website](#). Also it can be downloaded as a [zip archive from the website](#).

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, **Extensions** tab, **Extensions bundle** field.

- as an individual extension - associated with your document type in the **Document type** dialog, **Extensions** tab, **Individual extensions** section, **XML node renderer customizer** field.

Sample project reference

-  **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download [on the Oxygen XML Editor plugin website](#).
-  **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 Default Frameworks project, included in the *Oxygen Author SDK zip* available for download [on the Oxygen XML Editor plugin website](#).
-  **Note:** The Javadoc documentation of the Author API used in the example files is [available on the Oxygen XML Editor plugin website](#). Also it can be downloaded as a *zip archive from the website*.

Customizing the Default CSS of a Document Type

The easiest way of customizing the default CSS stylesheet of a document type is to create a new CSS stylesheet in the same folder as the customized one, import the customized CSS stylesheet and set the new stylesheet as the default CSS of the document type. For example let us customize the default CSS for DITA documents by changing the background color of the *task* and *topic* elements to red.

1. First you create a new CSS stylesheet called `my_dita.css` in the folder `${frameworks}/dita/css_classed` where the default stylesheet called `dita.css` is located. `${frameworks}` is the subfolder `frameworks` of the Oxygen XML Editor plugin Editor. 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 first open the Document Type Association preferences panel from menu **Options > Preferences > Document Type Association**. Select the DITA document type and start editing it by pressing the Edit button. In the Author tab of the document type edit dialog change the URI of the default CSS stylesheet from `${frameworks}/dita/css_classed/dita.css` to `${frameworks}/dita/css_classed/my_dita.css`.

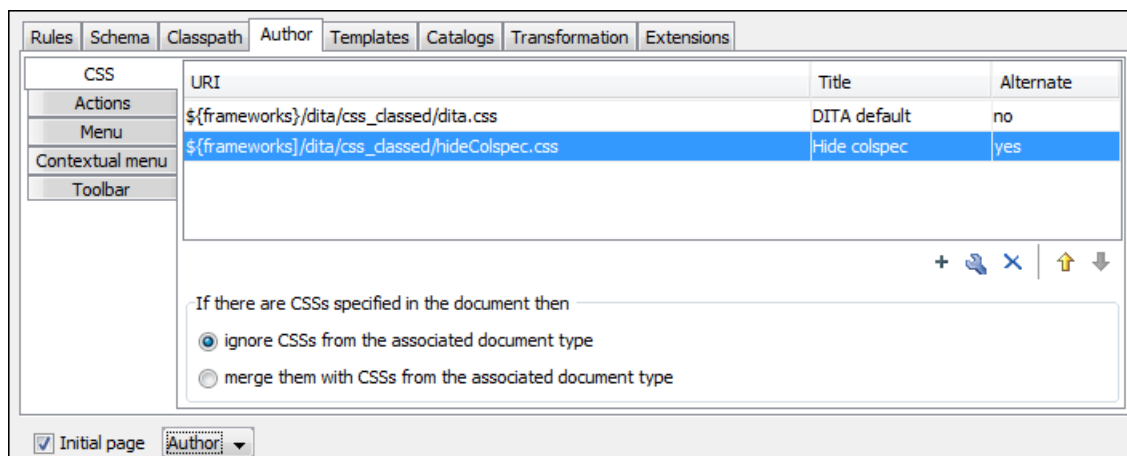


Figure 259: Set the location of the default CSS stylesheet

- Press OK in all the dialogs to validate the changes. Now you can start editing DITA documents based on the new CSS stylesheet. You can edit the new CSS stylesheet itself at any time and see the effects on rendering DITA XML documents in the **Author** mode by running the *Refresh* action available on the Author toolbar and on the DITA menu.

Document Type Sharing

Oxygen XML Editor plugin 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_INSTALL_DIR/frameworks` directory.



Important: For this approach to work, have the application installed to a folder with full write access.

Please follow these steps:

- Go to `OXYGEN_INSTALL_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_INSTALL_DIR/frameworks/docbook` as an example.
- Create your custom document type and save it externally, in the `custom_framework` directory.
- Configure the custom document type according to your needs, take special care to make all file references relative to the `OXYGEN_INSTALL_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.
- If everything went fine then you should have a new configuration file saved in: `OXYGEN_INSTALL_DIR/frameworks/custom_framework/custom_framework` after the Preferences are saved.
- Then, to share the new framework directory with other users, have them copy it to their `OXYGEN_INSTALL_DIR/frameworks` directory. The new document type will be available in the list of Document Types when Oxygen XML Editor plugin 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.

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>

CSS Support in Author

Author editing mode supports most CSS 2.1 selectors, a lot of CSS 2.1 properties and a few CSS 3 selectors. Also some custom functions and properties that extend the W3C CSS specification and are useful for URL and string manipulation are available to the developer who creates an Author editing framework.

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 to the *XML / XML Catalog* preferences page your own XML catalog file which maps the location to a custom CSS:

```
<uri name="http://host/path/to/location/custom.css" uri="path/to/custom.css"/>
```

In addition to this, you can also add in your XML Catalog file the following mapping:

```
<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 CSSs used to render the opened XML document in the visual **Author** editing mode. In this way your custom CSS can be used without the need to modify all other CSSs contributed in the document type configuration.

Media Type oxygen

The CSS stylesheets can specify how a document is presented on different media: on the screen, on paper, on speech synthesizer, etc. You can specify that some of the selectors from your CSS should be taken into account only in the **Author** mode and ignored in the rest. This can be accomplished by using the oxygen media type.

For instance using the following CSS:

```
b{
  font-weight:bold;
  display:inline;
}

@media oxygen{
  b{
    text-decoration:underline;
  }
}
```

would make a text bold if the document was opened in a web browser that does not recognize `@media oxygen` and bold and underlined in Oxygen XML Editor plugin Author.


You can use this media type to group specific Oxygen XML Editor plugin CSS selectors and have them ignored when opening the documents with other viewers.

Standard W3C CSS Supported Features

oXygen 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
*	Universal selector	CSS Level 2	Matches any element
E	Type selector	CSS Level 2	Matches any E element (i. e. an element with the local name E)
E F	Descendant selector	CSS Level 2	Matches any F element that is a descendant of an E element.

Expression	Name	CSS Level	Description / Example
<code>E > F</code>	Child selectors	CSS Level 2	Matches any <code>F</code> element that is a child of an element <code>E</code> .
<code>E:lang(c)</code>	Language pseudo-class	CSS Level 2	Matches element of type <code>E</code> if it is in (human) language <code>c</code> (the document language specifies how language is determined).
<code>E + F</code>	Adjacent selector	CSS Level 2	Matches any <code>F</code> element immediately preceded by a sibling element <code>E</code> .
<code>E[foo]</code>	Attribute selector	CSS Level 2	Matches any <code>E</code> element with the " <code>foo</code> " attribute set (whatever the value).
<code>E[foo="warning"]</code>	Attribute selector with value	CSS Level 2	Matches any <code>E</code> element whose " <code>foo</code> " attribute value is exactly equal to " <code>warning</code> ".
<code>E[foo~="warning"]</code>	Attribute selector containing value	CSS Level 2	Matches any <code>E</code> element whose " <code>foo</code> " attribute value is a list of space-separated values, one of which is exactly equal to " <code>warning</code> ".
<code>E[lang ="en"]</code>	Attribute selector containing hyphen separated values	CSS Level 2	Matches any <code>E</code> element whose " <code>lang</code> " attribute has a hyphen-separated list of values beginning (from the left) with " <code>en</code> ".
<code>E:before</code> and <code>E:after</code>	Pseudo elements	CSS Level 2	The <code>:before</code> and <code>:after</code> pseudo-elements can be used to insert generated content before or after an element's content.
<code>E:first-child</code>	The first-child pseudo-class	CSS Level 2	Matches element <code>E</code> when <code>E</code> is the first child of its parent.
<code>E:not(s)</code>	Negation pseudo-class	CSS Level 2	An <code>E</code> element that does not match simple selector <code>s</code> .
<code>E:hover</code>	The hover pseudo-class	CSS Level 2	The <code>:hover</code> 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.
<code>E:focus</code>	The focus pseudo-class	CSS Level 2	The <code>:focus</code> pseudo-class applies while an element has the focus (accepts keyboard input).
<code>E#myid</code>	The ID selector	CSS Level 2	Matches any <code>E</code> element with ID equal to " <code>myid</code> ".  Important: Limitation: In oXygen the match is performed taking into account only the attributes with the exact name: " <code>id</code> ".

Expression	Name	CSS Level	Description / Example
<code>E[att^="val"]</code>	Substring matching attribute selector	CSS Level 3	An E element whose <code>att</code> attribute value begins exactly with the string <code>val</code> .
<code>E[att\$="val"]</code>	Substring matching attribute selector	CSS Level 3	An E element whose <code>att</code> attribute value ends exactly with the string <code>val</code> .
<code>E[att*="val"]</code>	Substring matching attribute selector	CSS Level 3	An E element whose <code>att</code> attribute value contains the substring <code>val</code> .
<code>E:root</code>	Root pseudo-class	CSS Level 3	Matches the root element of the document. In HTML, the root element is always the HTML element.
<code>E:empty</code>	Empty pseudo-class	CSS Level 3	An E element which has no text or child elements.
<code>E:nth-child(n)</code>	The nth-child pseudo-class	CSS Level 3	An E element, the n-th child of its parent.
<code>E:nth-last-child(n)</code>	The nth-last-child pseudo-class	CSS Level 3	An E element, the n-th child of its parent, counting from the last one.
<code>E:nth-of-type(n)</code>	The nth-of-type pseudo-class	CSS Level 3	An E element, the n-th sibling of its type.
<code>E:nth-last-of-type(n)</code>	The nth-last-of-type pseudo-class	CSS Level 3	An E element, the n-th sibling of its type, counting from the last one.
<code>E:last-child</code>	The last-child pseudo-class	CSS Level 3	An E element, last child of its parent.
<code>E:first-of-type</code>	The first-of-type pseudo-class	CSS Level 3	An E element, first sibling of its type.
<code>E:last-of-type</code>	The last-of-type pseudo-class	CSS Level 3	An E element, last sibling of its type.
<code>E:only-child</code>	The only-child pseudo-class	CSS Level 3	An E element, only child of its parent.
<code>E:only-of-type</code>	The only-of-type pseudo-class	CSS Level 3	An E element, only sibling of its type.
<code>ns E</code>	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: <pre>@namespace ns "http://some_namespace_uri";</pre> See Namespace Selector on page 460.
<code>E!>F</code>	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 461.

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 plugin 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 <code>http://sync.example.org</code> 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 <code>http://example.com/foo</code> namespace.

Defining only prefixed namespaces

Given the namespace declaration:

```
@namespace sync "http://sync.example.org";
```

Then:

sync A	represents the name A in the <code>http://sync.example.org</code> 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 <code>http://www.oxygenxml.com/sample/documentation/abstracts</code> namespace.
abs def:before	represents the <code>:before</code> pseudo-element of the "def" element from the <code>http://www.oxygenxml.com/sample/documentation/abstracts</code> namespace.

Subject Selector

Oxygen XML Editor plugin Author supports the subject selector described in CSS Level 4 (currently a working draft at W3C <http://dev.w3.org/csswg/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 plugin validates all CSS 2.1 properties, but does not render in Author mode *aural* and *paged* categories properties, as well as some of the values of the *visual* category, listed below under the **Ignored Values** column. For the Oxygen XML Editor plugin specific (extension) CSS properties, go to [Oxygen CSS Extensions](#) on page 469.

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	
'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	

Name	Rendered Values	Ignored Values
'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	
'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

Name	Rendered Values	Ignored Values
'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'		ALL
'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 -oxy-lower-cyrillic-uk -oxy-upper-cyrillic-ru -oxy-upper-cyrillic-uk box diamond check hyphen none inherit	lower-greek armenian georgian
'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	

Name	Rendered Values	Ignored Values
'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'		ALL
'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'		ALL
'quotes'		ALL
'right'		ALL
'table-layout'	auto	fixed inherit
'text-align'	left right center inherit	justify
'text-decoration'	none [underline overline line-through] inherit	blink
'text-indent'		ALL
'text-transform'	ALL	
'top'		ALL
'unicode-bidi'	bidi-override normal embed inherit	
'vertical-align'	baseline sub super top text-top middle bottom text-bottom inherit	<percentage> <length>

Name	Rendered Values	Ignored Values
'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 plugin 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 plugin 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 plugin 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 plugin 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 <i>font-size</i> of the relevant font.
ex	The value must be interpreted as a size. 1 ex is equal to the <i>height</i> 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 supports some of the at-rules specified by CSS Level 2.1 and 3.

The @font-face at-rule

Oxygen XML Editor plugin 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 specified font-family must match the name of the font declared in the .ttf file.

The @media at-rule

Oxygen XML Editor plugin supports several media types, allowing you to set different styles for presenting a document on different media (on the screen, on paper and so on). The following media types are supported:

- screen - the styles marked with this media type are used only for rendering a document in the **Author** mode;
- all - the styles marked with this media type are used for rendering a document in the **Author** mode and also for printing the document;
- oxygen - the styles marked with this media type are used only for rendering a document in the **Author** mode;



Note: This is an Oxygen XML Editor plugin specific media.

- print - the styles marked with this media type are used only for printing a document.

Oxygen CSS Extensions

CSS stylesheets provide support mainly for displaying documents. When editing documents some non-standard, oXygen specific CSS extensions are useful, for example:

- property for marking foldable elements in large files;
- enforcing a display mode for the XML tags regardless of the current mode selected by the author user;
- construct an URL from a relative path location;
- string processing functions.

Additional CSS Selectors

Oxygen XML Editor plugin 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, go to **Options > Preferences > 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 CSSs, 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* are expanded by default in Author mode and *the referred content is displayed*. The referred resources are loaded and displayed inside the element or entity that refers them.
 - You can use the *reference* property to customize the way these references are rendered in the **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 referred, 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 plugin 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 plugin 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 plugin 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:

- **Ctrl (Meta on Mac OS) + NumPad +/- > Document > Folding >  Close Other Folds > Ctrl (Meta on Mac OS) + NumPad +/-** - Folds all the elements except the current element.
- **Document > Folding >  Collapse Child Folds (Ctrl+Decimal) (Ctrl+NumPad+/-) ((Cmd+NumPad+/- on Mac OS))** - Folds the elements indented with one level inside the current element.
- **Document > Folding >  Expand Child Folds (Ctrl+NumPad++) ((Cmd+NumPad++))** - Unfolds all child elements of the currently selected element.
- **Document > Folding >  Expand All (Ctrl+NumPad+*) ((Cmd+NumPad+* on Mac OS))** - Unfolds all elements in the current document.
- **Document > Folding >  Toggle Fold (Alt+Shift+Y) ((Cmd+Alt+Y on Mac OS))** - 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 elements that are kept visible. It accepts as value an element name or a list of comma separated element names. 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 plugin Author displays the element name as pseudo-content for empty elements, if the <http://www.oxygenxml.com/doc/ug-editor/topics/preferences-author.html#Show placeholders for empty elements> option is enabled and there is no `before` or `after` content set is 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 is 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 plugin 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 plugin 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 plugin 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 plugin 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 plugin Author allows you to declare some elements to be *links*. This is especially useful when working with many documents which refer each other. The links allow for an easy way to get from one document to another. Clicking on the link marker will open the referred resource in an editor.

To define the element which should be considered a link, you must use the property `link` 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 plugin 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 plugin does not display element tags. You can use this 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 plugin uses all these parameters to construct an absolute location.

```
oxy_url ( location , loc_1 , loc_2 )
```

location	The location as string. If not absolute, will be solved relative to the CSS file URL.
loc_1 ... loc_n	Relative location path components as string. (optional)

The following function:

```
oxy_url('http://www.oxygenxml.com/css/test.css', '../dir1/', 'dir2/dir3/',
'../../../../dir4/dir5/test.xml')
```

returns

```
'http://www.oxygenxml.com/dir1/dir4/dir5/test.xml'
```

As a concrete example 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_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

This 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**:

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 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 in PERL syntax, 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 referred 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 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.

The form control allows you to edit Processing Instructions (PIs), the value of an attribute, or the text content of an element. This is specified using the `edit` property. This property can have the following values:

- **@attribute_name** - specifies that the presented/edited value is the value of an attribute;
- **#text** - specifies that the presented/edited value is the simple text value of an element. This text can contain built-in character entities.



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.

You can use a number of built-in form controls through the `type` property. The following values are recognized:

- **text** - a text field with optional content completion capabilities is used to present and edit a value;
- **combo** - a combo-box is used to present and edit a value;
- **check** - a single check box or multiple check boxes are used to present and edit a value;
- **popupSelection** - a pop-up with single/multiple selection is used as form control;
- **button** - a button that invokes an author action is used as form control;
- **urlChooser** - a text field with a browse button is used as form control.
- **datePicker** - a text field with a calendar browser button is used as form control.

To watch our video demonstration about form controls, go to http://oxygenxml.com/demo/Form_Controls.html.

The Text Field Form Control

A text field with optional content completion capabilities is used to present and edit the value of an attribute or an element. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_textfield`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Text form control, its value has to be `text`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```



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;
- `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 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. In case 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 are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `#{comma}` variable;
- `tooltip` - specifies a tooltip for the form control itself. This tooltip is displayed when you hover the form control using your cursor;

- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.

Text Field Form Control

```
element {
  content: "Label: "
  oxy_editor(
    type, text,
    edit, "@my_attr",
    values, "value1, value2"
    columns, 40);
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_textfield` dedicated function.

```
element {
  content: "Label: "
  oxy_textfield(
    edit, "@my_attr",
    values, "value1, value2"
    columns, 40);
}
```

The Combo Box Form Control

A combo box is used to present and edit the value of an attribute or an element. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_combobox`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Combo box form control, its value has to be `combo`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```

- **#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;
- `visible` - specifies whether 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. The **true** value generates an editable combo-box that allows you to insert other values than the proposed ones. The **false** value generates a combo-box that only accepts the proposed values;
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable;
- `values` - specifies the values that populate the content completion list of proposals. In case 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`. 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 available only for read-only Combo boxes (Combo boxes that have the `editable` property set to `false`).

- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.

Combo Box Form Control

```
comboBox:before {
  content: "A combo box that edits an attribute value. The possible values are provided from
  CSS:"
  oxy_editor(
    type, combo,
    edit, "@attribute",
    editable, true,
    values, "value1, value2, value3",
    labels, "Value no1, Value no2, Value no3");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_combobox` dedicated function.

```
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");
}
```

The Check Box Form Control

A single check-box or multiple check-boxes are used to present and edit the value on an attribute or element. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_checkbox`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Check Box form control, its value has to be `checkbox`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, checkbox,
    edit, "@fc:my_attr"
  )
}
```



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` - in case 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 are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable;
- `visible` - specifies whether 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. In case 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`. 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 the 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. In case this property is not specified, the **values** property is used as 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. In case the value of the **color** property is **inherit**, the form control has the same color as the element in which it is inserted.

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_editor(
  type, check,
  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_editor(
  type, check,
  edit, "@attribute",
  values, "true, yes, on",
  uncheckedValues, "false, no, off",
  resultSeparator, ",",
  labels, "Present, Working, Started");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_checkbox` dedicated function.

```
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");
}
```

The Pop-up Form Control

A pop-up with single or multiple selection is used as a form control. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_popup`. This type of form control supports the following properties:

- **type** - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Pop-up form control, its value has to be `popupSelection`;
- **edit** - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```

- **#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: In case 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. In case 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 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 are a list of tooltips separated by commas. In case 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. In case these values are not specified in the CSS, they are collected from the associated XML Schema;
- `resultSeparator` - in case 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 of multiple values. The predefined values of this property are `single` and `multiple`;
- `labels` - specifies the label associated with each entry used for presentation. In case this property is not specified, the `values` property is used as label;
- `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;
- `renderSort` - allows you to sort the values rendered on the pop-up form control label. The possible values of this property are `ascending` and `descending`;
- `editorSort` - allows you to sort the values rendered on the pop-up window. The possible values of this property are `ascending` and `descending`;
- `renderSeparator` - 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`. To make the Pop-up form control inherit its font from its parent element, set the `fontInherit` property to `true`;



Tip: In the below example, the value of the `fontInherit` property is **true**, meaning that the pop-up form control inherits the font size of `30px` from the `font-size` property.

Pop-up Form Control

```
popupWithMultipleSelection:before {
  content: " This editor edits an attribute value. The possible values are      specified
  inside the CSS: "
  oxy_editor(
    type, popupSelection,
    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;
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_popup` dedicated function.

```
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;
}
```

The Button Form Control

This form control contributes a button that invokes a *custom Author action* (defined in the associated Document Type) using its defined ID. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_button`. The following properties are supported:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Button form control the value of the `type` property is `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. In case 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. In case 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 *Author actions*, that is invoked when you click the button;



Note: The element that contains the `Button` form control represents the context where the action is invoked.

- `visible` - specifies whether 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 only the icon 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 only the icon 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);
}
```

Button Form Control

```
button:before {
  content: "Label:"
  oxy_editor(
    type, button,
    /* This action is declared in the document type associated with the XML document. */
    actionID, "insert.popupWithMultipleSelection");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_button` dedicated function.

```
button:before {
  content: "Label:"
  oxy_button(
    /* This action is declared in the document type associated with the XML document. */
    actionID, "insert.popupWithMultipleSelection");
}
```

The Button Group Form Control

A pop-up menu is shown, which can invoke one of the several custom Author actions (defined in the associated Document Type) specified by their ID. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_buttonGroup`. This type of form control supports the following properties:

- `actionIDs` - comma separated IDs of the actions to be displayed in the pop-up menu;

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control
- `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. In case the caret is not inside the element that holds the form control, the `element` value is selected automatically;
- `visible` - specifies whether 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 pop-up menu. The values of this property can be `text`, `and icon`, or `both`;
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `#{comma}` variable;
- `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`. To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.

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");
}
```

The Text Area Form Control

A text area with optional syntax highlight capabilities is used to present and edit the value of an attribute or an element. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_textArea`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Text Area form control, its value has to be `textArea`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```

- **#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;
- `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 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. In case the form control has more lines, you are able to scroll and see them all;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```

- **#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

- `contentType` - specifies the type of content for which the form control offers syntax highlight. 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 white spaces and new lines of the value are preserved and edited. If the `rows` and `columns` properties are not specified, the Text Area 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 have to be specified. In case these are not specified, the form control considers the value to be `pre`.

- `normal` - the white spaces and new lines are normalized.

The following example presents a text area with CSS syntax highlight which calculates its own dimension, and a second one with XML syntax highlight 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);
}
```


The URL Chooser Form Control

A field that allows you to select local and remote resources is used as a form control. The inserted reference will be made relative to the current opened editor's URL. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_urlChooser`. This type of editor supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the URL Chooser editor, its value has to be `urlChooser`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```

- **#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;
- `color` - specifies the foreground color of the form control. In case 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 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`.

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_editor(
    type, urlChooser,
    edit, "@file",
    columns 25);
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_urlChooser` dedicated function.

```
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);
}
```

The Date Picker Form Control

A text field with a calendar browser is used as a form control. The browse button shows a date chooser allowing you to easily choose a certain date. It can be added using the generic function `oxy_editor`. Alternatively, you can use the dedicated function: `oxy_datePicker`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Date picker form control, its value has to be `datePicker`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). 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:

```
@namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
myElement {
  content: oxy_editor(
    type, combo,
    edit, "@fc:my_attr"
  )
}
```

- **#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;
- `color` - specifies the foreground color of the form control. In case 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 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. In case you introduce a date that does not respect the format, the `datePicker` form control is rendered with red foreground. By default, the input is validated. To disable the validation, set this property to `false`.

Date Picker Form Control

```
date {
  content:
    oxy_label(text, "Date time attribute with format defined in CSS: ", width, 300px)
    oxy_editor(
      type, datePicker,
      columns, 16,
      edit, "@attribute",
      format, "yyyy-MM-dd");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_datePicker` dedicated function.

```
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");
}
```

Editing PIs Using Form Controls

Oxygen XML Editor plugin allows you to edit *processing instructions*, *comments*, and *cdata* 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;
}
```

Implementing Custom Form Controls

In case the built-in form controls are not enough, you can implement custom form controls in Java and 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** - in case 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.



Note: If the custom form control chooses to perform the commit by itself, it must do so after it triggers the `ro.sync.ecss.extensions.api.editor.InplaceEditingListener.editingStopped(EditingE notification`.

If the custom form control is intended to work in the Oxygen XML Editor plugin 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 way.

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 Author 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 , ",") ');
  )
}
```

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 horizontal space reserved for the content. The value of this property has the same format as the value of the CSS `width` property. In case this value is not specified, the text is wrapped;
- `color` - specifies the foreground color of the form control. In case 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. In case 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`.

```
element{
  content: oxy_label(text, "Label Text", styles,
    "font-size:2em;color:red;");
}
```

If the text from an `oxy_label()` function contains new lines, for example `oxy_label(text, 'LINE1\nLINE2', width, 100px)`, the text is split and the new line has the specified width (100 pixels in this case).



Note: The text is split only after `\A`. In case a `width` is specified for the `oxy_label()` function and a `\A` is encountered, the new line has the specified width.

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)
}
```

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();
}
```

Arithmetic Functions

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 param1 to paramN.
<code>oxy_subtract(param1, ..., paramN, 'returnType')</code>	Subtracts the values of parameters param2 to paramN from param1.
<code>oxy_multiply(param1, ..., paramN, 'returnType')</code>	Multiplies the values of parameters from param1 to paramN.
<code>oxy_divide(param1, param2, 'returnType')</code>	Performs the division of param1 to param2.
<code>oxy_modulo(param1, param2, 'returnType')</code>	Returns the remainder of the division of param1 to param2.



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 they are used to hide/show the colspec elements from CALS tables. To take a look at the implementation, see:

1. `OXYGEN_INSTALL_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](#) can be used directly in your framework ("Author/Actions") to work with custom pseudo classes:

1. [TogglePseudoClassOperation](#)
2. [SetPseudoClassOperation](#)
3. [RemovePseudoClassOperation](#)

Builtin CSS Stylesheet

When Oxygen XML Editor plugin 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 plugin

```

@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='rlo'] { 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;
}
```

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 [Author SDK distribution](#) in the "oxygenAuthorSDK\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: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:sequence>
    <xs:attribute name="width" type="xs:string"/>
  </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>
</xs:schema>

```

abs.xsd

This sample file can also be found in the *Author SDK distribution* in the "oxygenAuthorSDK\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 *Author SDK distribution* in the oxygenAuthorSDK\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 [Author SDK distribution](#) in the "oxygenAuthorSDK\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>
                <thead>
                    <tr>
                        <th></th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <th></th>
                        <td></td>
                    </tr>
                </tbody>
            </table>
        </section>
    </section>

```

```

        <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>
element.</td>
    </tr>
    <tr>
      <td>
        <b>for-each</b>
      </td>
      <td>The xsl:for-each element causes
iteration over the nodes selected by
a node-set expression.</td>
    </tr>
    <tr>
      <td colspan="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>
    <thead>
      <tr>
        <th>Function</th>
        <th>Description</th>
      </thead>
    </thead>
    <tbody>
      <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>
    </tbody>
  </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 [Author SDK distribution](#) in the "oxygenAuthorSDK\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 plugin website: <http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-startup-project.zip>

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 development work 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 Author 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 or server: http://www.oxygenxml.com/license_server.html.

```
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 to the end user. 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 plugin 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 506.

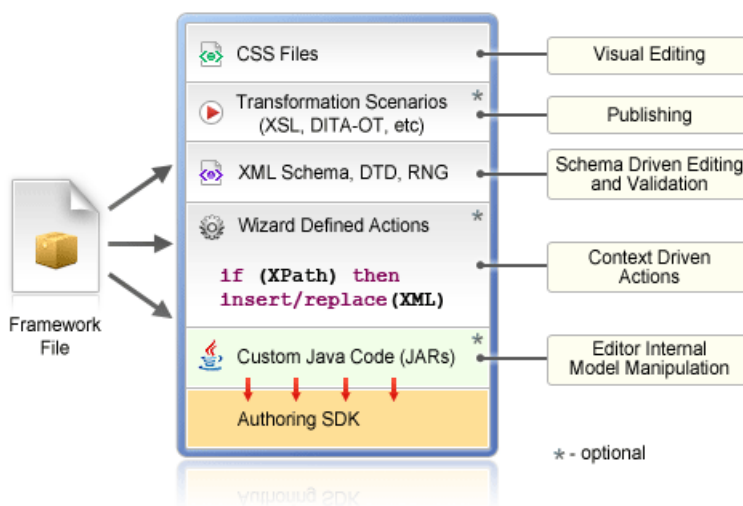
Running the Author component embedded in a third-party Java/Swing application requires:

- Oracle (Sun) Java JRE version 1.6 or newer;
- At least 100 MB disk space and 100MB free memory;

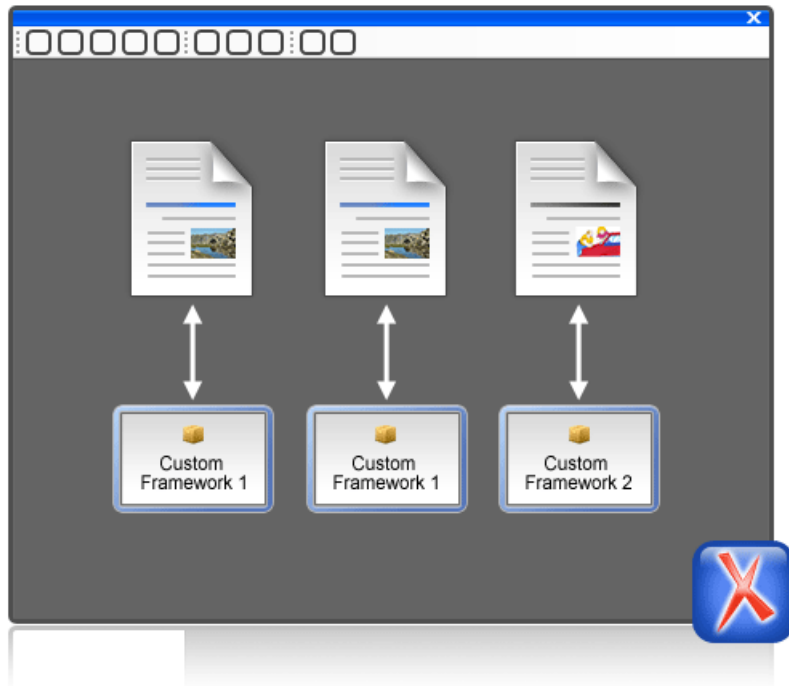
Customization

For a special type of XML, you can create a custom framework (which also works in an Oxygen standalone version). Oxygen XML Editor plugin already has frameworks for editing DocBook, DITA, TEI, and so on. Their sources are available in [the Author 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 plugin 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.

Packing a fixed set of options

The Author Component shares a common internal architecture with the standalone application although it does not have a **Preferences** dialog. But the Author Component Applet can be configured to use a fixed set of user options on startup.

The sample project contains a resource called `APPLET_PROJECT/resources/options.zip.jar`. The JAR contains a ZIP archive which contains a file called `options.xml`. Such an XML file can be obtained by exporting to an XML format from a standalone application.

To create an *options file* in the Oxygen XML Editor plugin:

- make sure the options that you want to set are not ;
- 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 in the [sample project](#) in the `lib/apiSrc.zip` archive. The sample project also 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:

- Unpack the sample project archive and look for Java sources of the sample Applet implementation. 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 `author-component-dita.html` and `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 `author-component-dita.jnlp` JNLP file can be edited to add more libraries. The packed frameworks and options are delivered using the JNLP file as JAR archives:

```
<jar href="resources/frameworks.zip.jar"/>
<jar href="resources/options.zip.jar"/>
```

- The sample frameworks and options JAR archives can be found in the `resources` directory.
- Use the `build.xml` ANT build file to pack the component. The resulting applet distribution is copied in the `dist` directory. From this on, you can copy the applet files on your web server.

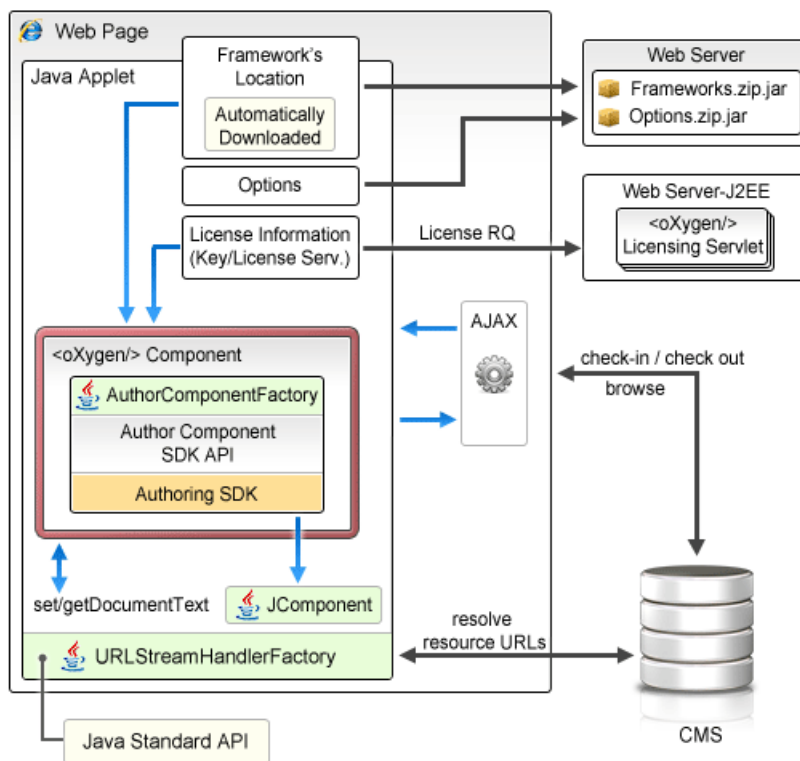


Figure 260: Oxygen XML Editor plugin Author Component deployed as a Java applet

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.

1. 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.

2. Generate a self-signed certificate.

Invoke the following in a command line terminal:

```
keytool -selfcert -alias myAlias -keystore keystore.pkcs -storetype PKCS12
```

3. 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.pkcs -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.pkcs` file (`keystore` parameter), keystore type (`storetype` parameter, with `JSK` or `PKCS12` as possible values), alias (`alias` parameter) and password (`password` parameter).
5. Sign the jar files using the certificate by running the `sign` Ant task available in [the applet project](#).

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
Windows XP	Passed	Passed	-	-	-	Passed	-	Passed	Passed
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

Using the Java 1.6 [LiveConnect](#) technology, applets can communicate with Javascript code which runs in the Web Page. Javascript code can call an applet's Java methods and from the Java code you can invoke Javascript code from the web page.

You are not limited to displaying only Swing dialogs from the applet. From an applet's operations you can invoke Javascript API which shows a web page and then obtains the data which has been filled 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 MAC 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 `URLConnectionHandlerFactory` 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.setURLConnectionHandlerFactory(new URLURLConnectionHandlerFactory() {
    public URLURLConnectionHandler createURLConnectionHandler(String protocol) {
        if("http".equals(protocol) || "https".equals(protocol)) {
            return new URLURLConnectionHandler() {
                @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

In the `author-component-dita.jnlp` JNLP file, refer additional libraries necessary for the JEuclid library to parse MathML equations:

```
<jar href="lib/jcip-annotations.jar"/>
<jar href="lib/jeuclid-core.jar"/>
<jar href="lib/batik-all-1.7.jar"/>
<jar href="lib/commons-io-1.3.1.jar"/>
<jar href="lib/commons-logging-1.0.4.jar"/>
<jar href="lib/xmlgraphics-commons-1.4.jar"/>
```

Copy these additional libraries to the component project `lib` directory from an `OXYGEN_INSTALLATION_DIRECTORY/lib` directory.

To edit specialized DITA Composite with MathML content, include the entire `OXYGEN_INSTALLATION_DIRECTORY/frameworks/mathml2` Mathml2 framework directory in the frameworks bundled with the component `frameworks.zip.jar`. This directory is used to solve references to MathML DTDs.

Adding MathML support using MathFlow

In the `author-component-dita.jnlp` JNLP file, refer additional libraries necessary for the MathFlow library to parse MathML equations:

```
<jar href="lib/MFComposer.jar"/>
<jar href="lib/MFExtraSymFonts.jar"/>
<jar href="lib/MFSimpleEditor.jar"/>
<jar href="lib/MFStructureEditor.jar"/>
<jar href="lib/MFStyleEditor.jar"/>
```

Copy these additional libraries from the MathFlow SDK.

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_INSTALLATION_DIRECTORY/frameworks/mathml2` Mathml2 framework directory in the frameworks bundled with the component `frameworks.zip.jar`. This directory is used to solve references to MathML DTDs.

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 14.2 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. Go to the installation directory of Oxygen XML and from the lib directory copy the httpclient-4.2.1.jar, httpcore-4.2.1.jar, commons-logging-1.1.1.jar and commons-codec-1.6.jar libraries. These libraries are used in the class path of the component (applet).

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 content that is bundled to form the `frameworks.zip.jar` must contain the additional plugin directories, besides the framework directories. The content must also contain a `plugin.dtd` file.



Note:

Copy the `plugin.dtd` file from an `OXYGEN_INSTALL_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 on the **<Oxygen/> XML Editor** website: <http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-startup-project.zip>

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="authorComponentAppletName" 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. This project contains the Author SDK and the basic resources to get started.

The project is available at <http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-startup-project.zip>.

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. Unpack the sample project archive 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.

2. Look inside `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. To add more libraries to your applet, edit the `author-component-dita.jnlp` JNLP file. The packed frameworks and options are delivered using the JNLP file as JAR archives:

```
<jar href="resources/frameworks.zip.jar"/>
<jar href="resources/options.zip.jar"/>
```

The sample frameworks and options JAR archives are located in the `resources` directory.



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 `build.xml` ANT build file to pack the component. The resulting applet distribution is copied in the `dist` 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:

1. Configure a new connection to your SharePoint site in the **Data Source Explorer** View.

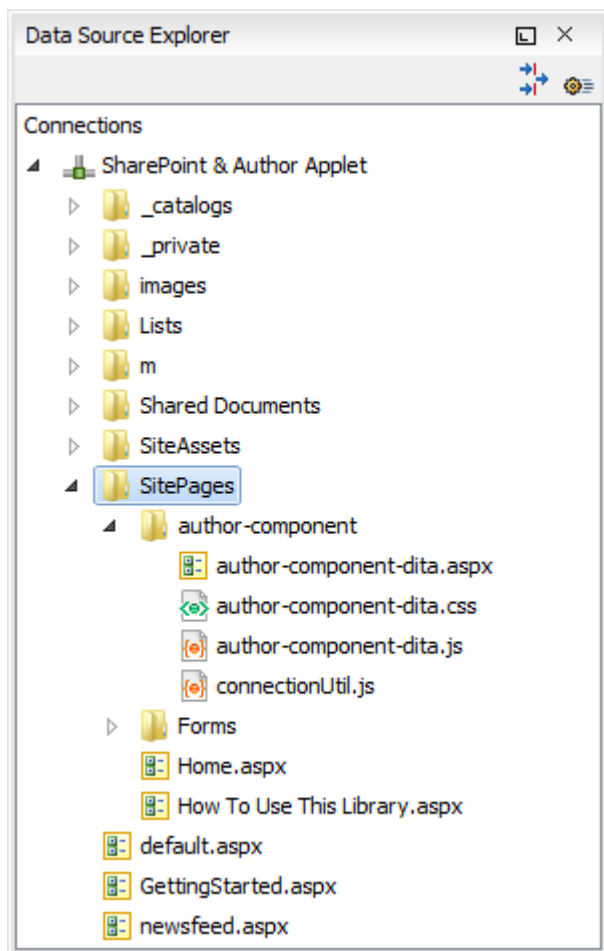


Note: To watch our video demonstration about connecting to repository located on a SharePoint server, go to http://www.oxygenxml.com/demo/SharePoint_Support.html.

2. Browse your new SharePoint connection site and select the **SitePages** folder;

3. Create a folder named `author-component` using the **New Folder** contextual menu action;

4. 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:

1. Navigate to the home page of your SharePoint site where you want to add the Author Component Java applet;
2. Select the **Page** tab from the ribbon located at top of the page and click the **Edit** button;
3. Select the **Insert** tab and click **Web Part**;
4. In the **Categories** panel, select **Media and Content**;
5. 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}")
```

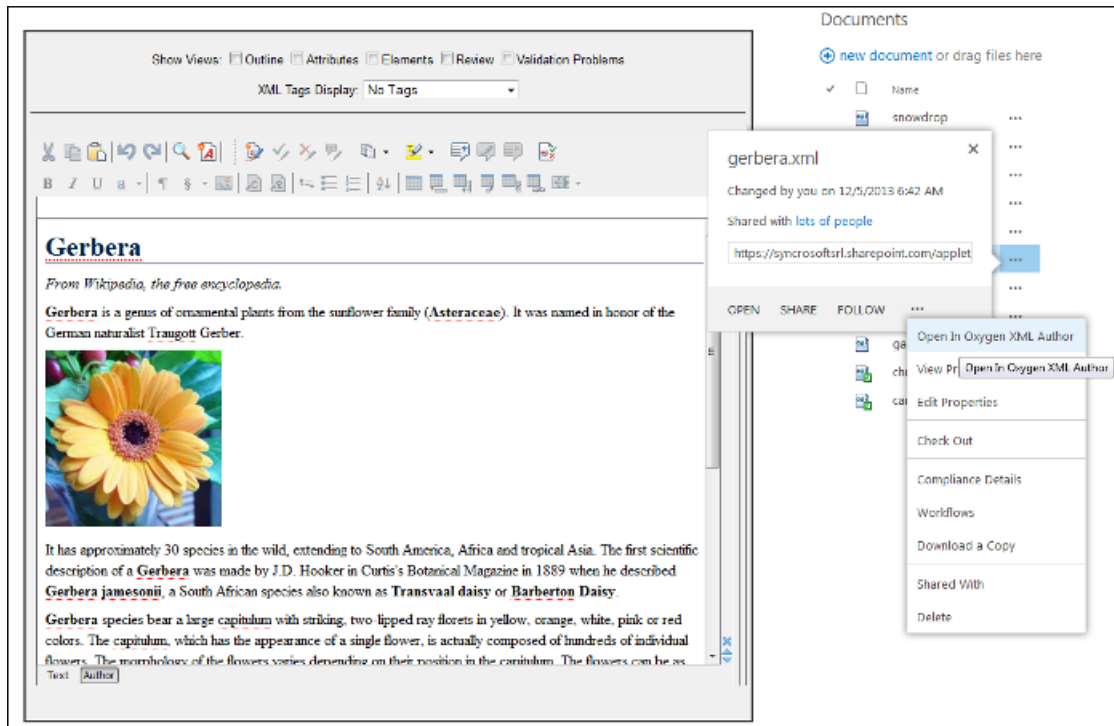


Note: This translates to a call to the `openInAuthor(itemUrl)` JavaScript function defined in the HTML fragment that was embedded in the Script Editor Web Part. The `{ItemUrl}` parameter will be expanded to the URL of the list item that the action is invoked on.

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

1. 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.

2. 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.

3. 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.

4. 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.

5. 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 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.

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 Author 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 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 Author SDK](#).

More than one framework can coexist in the same Oxygen 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 savviness). 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 writers.

- *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's 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 dialogs 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 plugin installation standalone (Author or Editor) comes with a main `oxygen.jar` library located in the `OXYGEN_INSTALLATION_DIRECTORY`. 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_INSTALLATION_DIRECTORY`.
2. Add to the **Java Build Path**->**Libraries** tab all JAR libraries present in the `OXYGEN_INSTALLATION_DIRECTORY/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. Download and add to the Java build path the additional JUnit libraries `jfcunit.jar` and `junit.jar`.
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=MCwCFGNoEGJSeiC3XCYIyaIvJzHhGhhqAhrNRDpEu8RIWb8icCJO7HqFVP4++A\|\|=|\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\"
            \"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());
    }
}
```

Chapter 9

API Frequently Asked Questions (API FAQ)

Topics:

- [Difference Between a Document Type \(Framework\) and a Plugin Extension](#)
- [Dynamically Modify the Content Inserted by the Writer](#)
- [Split Paragraph on Enter \(Instead of Showing Content Completion List\)](#)
- [Impose Custom Options for Writers](#)
- [Highlight Content](#)
- [How Do I Add My Custom Actions to the Contextual Menu?](#)
- [Adding Custom Callouts](#)
- [Change the DOCTYPE of an Opened XML Document](#)
- [Customize the Default Application Icons for Toolbars/Menus](#)
- [Disable Context-Sensitive Menu Items for Custom Author Actions](#)
- [Dynamic Open File in Oxygen XML Editor plugin Distributed via JavaWebStart](#)
- [Change the Default Track Changes \(Review\) Author Name](#)
- [Multiple Rendering Modes for the Same Author Document](#)
- [Obtain a DOM Element from an AuthorNode or AuthorElement](#)
- [Print Document Within the Author Component](#)
- [Running XSLT or XQuery Transformations](#)
- [Use Different Rendering Styles for Entity References, Comments or Processing Instructions](#)
- [Insert an Element with all the Required Content](#)

This section contains answers to common questions regarding the Oxygen XML Editor plugin customisations using the [Author 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 XML Editor plugin SDK, register to the [oXygen-SDK mailing list](#).

- *Obtain the Current Selected Element Using the Author API*
- *Debugging a Plugin Using the Eclipse Workbench*
- *Debugging an SDK Extension Using the Eclipse Workbench*
- *Extending the Java Functionality of an Existing Framework (Document Type)*
- *Controlling XML Serialization in the Author Component*
- *How can I add a custom Outline view for editing XML documents in the Text mode?*
- *Dynamically Adding Form Controls Using a StylesFilter*
- *Modifying the XML content on open*

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 plugin.

For the Oxygen XML Editor pluginPlugins 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 Preferences->"Document Type Association" there is a "DITA" document type.

If you *edit that document type* in Oxygen XML Editor plugin 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.

We have a special chapter in our user manual which explains how such document types are constructed and modified:

<http://www.oxygenxml.com/doc/ug-oxygen/index.html?q=/doc/ug-oxygen/topics/author-devel-guide-intro.html>

If you look on disk in the:

```
OXYGEN_INSTALL_DIR\frameworks\dita
```

folder there is a file called `dita.framework`. That file gets updated when you edit a document type from the Oxygen Preferences. Then you can share that updated file with all users.

The same folder contains some JAR libraries. These libraries contain custom complex Java operations which are called when the user presses certain toolbar actions.

If you want to add a custom action this topic explains how:

<http://www.oxygenxml.com/doc/ug-oxygen/index.html?q=/doc/ug-oxygen/tasks/addCustomActionHowTo.html>

We have an Author SDK which contains the Java sources from all the DITA Java customizations:

http://www.oxygenxml.com/oxygen_sdk.html#XML_Editor_Authoring_SDK

Dynamically Modify the Content Inserted by the Writer

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.err.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 plugin 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 Writers

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 writers 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.exml.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|step|xmp|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");
                        }
                    });
                }
            }
        });
    }
}
```

```

    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_INSTALL_DIR\lib\oxygen.jar` or `OXYGEN_INSTALL_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_INSTALL_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 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 plugin Distributed via JavaWebStart

Question

How can we dynamically open a file in an Oxygen XML Editor plugin distributed via JWS?

Answer

The JWS packager ANT build file which comes with Oxygen 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"/>
```

Signing the JNLP file indeed means that it is impossible to automatically generate a JNLP file containing some dynamic arguments.

But the JNLP does not need to be signed. Indeed the user probably receives this information when launching the application but at least in this way you should be able to dynamically generate a JNLP file via a PHP script based on the URL which was clicked by the user.

The generated JNLP would then take as argument the URL which needs to be opened when Oxygen starts.

Maybe a different approach (more complicated though) would be to have the JNLP file signed and always refer 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 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 following Preferences page: [Editor / Edit modes / Author / Review](#).
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/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/AuthorReviewController.html#setReviewerAuthorName\(java.lang.String\)](http://www.oxygenxml.com/InstData/Editor/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 drop-down button which contains alternative CSS styles for the same document. To add an alternative CSS stylesheet go to Oxygen XML Editor plugin Preferences->Document Type Association page, select the document type associated with your documents and press `Edit`. In the Document Type dialog that appears go to "Author" tab, "CSS" tab and add there references to alternate CSS stylesheets.

For example, one of the alternate CSSs that we offer for DITA document type is located here:

```
OXYGEN_INSTALL_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, 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:

```
InputStream is = new org.xml.sax.InputSource(URLUtil.correct(new File("test/personal.xml")).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 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.

```
AuthorSchemaManager schemaManager = this.authorAccess.getDocumentController().getAuthorSchemaManager();
WhatElementsCanGoHereContext context =
schemaManager.createWhatElementsCanGoHereContext(this.authorAccess.getEditorAccess().getCaretOffset());
List<CIElement> possibleElementsAtCaretPosition = schemaManager.whatElementsCanGoHere(context);
loop: for (int i = 0; i < possibleElementsAtCaretPosition.size(); i++) {
    CIElement possibleElement = possibleElementsAtCaretPosition.get(i);
    List<CIAttribute> attrs = possibleElement.getAttributes();
    if(attrs != null) {
        for (int j = 0; j < attrs.size(); j++) {
            CIAttribute ciAttribute = attrs.get(j);
            if (ciAttribute.getName().equals("class")) {
                if (ciAttribute.getDefaultValue() != null
                    && ciAttribute.getDefaultValue().contains(" topic/image ")) {
                    //Found a CIElement for image
                    //Create a fragment for it. The fragment contains all required child elements already built.
                    AuthorDocumentFragment frag = schemaManager.createAuthorDocumentFragment(possibleElement);
                    //Now set the @href to it.
                    //Ask the user and obtain a value for the @href
                    //Then:

                    String href = "test.png";
                    List<AuthorNode> nodes = frag.getContentNodes();
                    if(!nodes.isEmpty()) {
                        AuthorElement imageEl = (AuthorElement) nodes.get(0);
                        imageEl.setAttribute("href", new AttrValue(href));
                    }
                    //And insert the fragment.

                    this.authorAccess.getDocumentController().insertFragment(this.authorAccess.getEditorAccess().getCaretOffset(),
                    frag);
                    break loop;
                }
            }
        }
    }
}
```

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 to a folder on your hard drive.



Note: Name the folder OXYGEN_DIR.

2. Download the *Plugins SDK*.
3. Create an Eclipse Java Project (let's call it MyPluginProject) with the Java sources from one of the sample plugins (the Workspace Access plugin for example).
4. In the Project root folder, create a folder called myPlugin and add the plugin.xml from the sample plugin in there. Modify the added plugin.xml to add a library reference to the project's classes directory: `<library name=" ../classes "/>`.
5. Copy the plugin.dtd from the OXYGEN_DIR/plugins folder in the root Project folder.
6. 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.
7. 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 -Dcom.oxygenxml.editor.plugins.dir=D:\projects\MyPluginProject

```

8. Add a break point in one of the source Java classes.
9. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

Debugging an 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. Download the *Author SDK*.
3. 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).
4. 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.
5. 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.
6. 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

```

7. Add a break point in one of the source Java classes.
8. 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. Download the Author SDK, create a Java project (we work with Eclipse for example) which adds to the classpath all libraries of the SDK.
2. Also add to the project's class path the: "OXYGEN_INSTALL_DIR\frameworks\docbook\docbook.jar".
3. Create a class which 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_INSTALL_DIR\frameworks\docbook\custom.jar".
6. Start Oxygen, in the **Preferences > Document Type Association->** page 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.
7. 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 plugin, 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 XPath's 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);
                        }
                    });
                }
            }
        }
    });
}

```

```

        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* on page 480. 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.
                // This will be a comma separated enumeration: France, Spain, Great Britain
                String countries = readCountriesFromFile();
                Map<String, Object> formControlArgs = new HashMap<String, Object>();
                formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_EDIT, "#text");
                formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_TYPE, InplaceEditorArgumentKeys.TYPE_COMBOBOX);
                formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_VALUES, countries);
                formControlArgs.put(InplaceEditorArgumentKeys.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("styles", "** {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;
    }
}
```

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)
```

Chapter 10

Transforming Documents

Topics:

- [Output Formats](#)
- [Transformation Scenario](#)
- [Using the Oxygen WebHelp Plugin](#)
- [XSLT Processors](#)
- [XSL-FO Processors](#)

XML is mainly used to store, carry, and exchange data. When you want to view the data in a more user friendly form, do one of the following:

- use an XML-compliant user agent;
- transform the XML document to a format that can be read by other user agents. This process is known as **Transformation**.

Output Formats

Within the current version of Oxygen XML Editor plugin you can transform your XML documents to the following formats without having to exit from the application:

- **PDF** - Adobe Portable Document Format (PDF) is a compact binary file format that can be viewed and printed by anyone, anywhere across a broad range of hardware and software using the free PDF Viewer from [Adobe](#).
- **PS** - PostScript is the leading printing technology from [Adobe](#) for high-quality, best-in-class printing solutions ranging from desktop devices to the most advanced digital presses, platemakers, and large format image setters in the world. PostScript files can be viewed using viewers such as GhostScript, but are more commonly created as a prepress format.
- **TXT** - Text files are Plain ASCII Text and can be opened in any text editor or word processor.
- **XML** - XML stands for eXtensible Markup Language and is a [W3C](#) standard markup language, much like HTML, which was designed to describe data. XML tags are not predefined in XML. You must define your own tags. XML uses a Document Type Definition (DTD), an XML Schema or a Relax NG schema to describe the data. XML with a DTD, XML Schema or Relax NG schema is designed to be self-descriptive. XML is not a replacement for HTML. XML and HTML were designed with different goals:
 - XML was designed to describe data and to focus on what data is.
 - HTML was designed to display data and to focus on how data looks.
 - HTML is about displaying information, XML is about describing information.
- **XHTML** - XHTML stands for eXtensible HyperText Markup Language, a [W3C](#) standard. XHTML is aimed to replace HTML. While almost identical to HTML 4.01, XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application.

For transformation to formats that are not listed above simply install the tool chain required to perform the transformation and process the xml files created with Oxygen XML Editor plugin in accordance with the processor instructions.

All formatting during a transformation is provided under the control of an Extensible Stylesheet (XSLT). Specifying the appropriate XSLT enables transformation to the above formats and preparation of output files for specific user agent viewing applications, including:

- **HTML** - HTML stands for Hyper Text Markup Language and is a [W3C Standard](#) for the World Wide Web. HTML is a text file containing small markup tags. The markup tags tell the Web browser how to display the page. An HTML file must have an `htm` or `html` file extension. An HTML file can be created using a simple text editor.
- **HTML Help** - [Microsoft HTML Help](#) is the standard help system for the Windows platform. Authors can use HTML Help to create online help for a software application or to create content for a multimedia title or Web site. Developers can use the HTML Help API to program a host application or hook up context-sensitive help to an application.
- **JavaHelp** - JavaHelp software is a full-featured, platform-independent, extensible help system from [Sun Microsystems/Oracle](#) that enables developers and authors to incorporate online help in applets, components, applications, operating systems, and devices. JavaHelp is a free product and the binaries for JavaHelp can be redistributed.
- **Eclipse Help** - Eclipse Help is the help system incorporated in the [Eclipse platform](#) that enables Eclipse plugin developers to incorporate online help in their plugins.

Many other target formats are possible, these are the most popular. The basic condition for transformation to any format is that your source document is well-formed. Always, make sure that the XSL used for the transformation is the right one according to the desired output format and with the input source definition. For example, if you want to transform to HTML format using a DocBook html stylesheet, your source XML document should respect the DocBook DTD.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an output document by using special formatting vocabulary.

XSL consists of three parts:

- **XSL Transformations (XSLT)** - XSLT is a language for transforming XML documents.

- **XML Path (XPath) Language** - XPath is an expression language used by XSLT to access or refer parts of an XML document. XPath is also used by the XML Linking specification.
- **XSL Formatting Objects (XSL:FO)** - XSL:FO is an XML vocabulary for specifying formatting semantics.

Oxygen XML Editor plugin supports XSLT/XPath version 1.0 using Saxon 6.5.5, Xalan, Xsltproc, MSXML (3.0, 4.0, .NET) and XSLT/XPath 2.0 by using Saxon 9.5.0.1 HE, Saxon 9.5.0.1 PE, and Saxon 9.5.0.1 EE. *The validation* is done also depending on the stylesheet version.

Transformation Scenario

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, and others) 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 plugin, define a transformation scenario to apply to that document. A scenario is a set of values for various parameters defining a transformation. It is not related to any particular document but to a document type:

- **Scenarios that apply to XML files** - Such a scenario contains the location of an XSLT stylesheet that is applied on the edited XML document and other transform parameters.
- **Scenarios that apply to XSLT files** - Such a scenario contains the location of an XML document that the edited XSLT stylesheet is applied on and other transform parameters.
- **Scenarios that apply to XQuery files** - Such a scenario contains the location of an XML source that the edited XQuery file is applied on and 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 like `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** - Such a scenario 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.
- **Scenarios that apply to XProc files** - Such a scenario contains the location of an XProc script and other transform parameters.
- **DITA-OT scenarios** - Such a scenario provides the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Editor plugin 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.



Note:

Status messages generated during the transformation process are displayed in the [Console 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 transform parameters and the location of an XSLT stylesheet that Oxygen XML Editor plugin applies 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 transform parameters and the location of an XQuery file that Oxygen XML Editor plugin applies 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 plugin 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 options and parameters of an ANT script;
- **XSLT transformation** - specifies transform parameters and the 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** - contains the location of an XProc script and other transform parameters;
- **XQuery transformation** - specifies transform parameters and the 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 like `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**;
- Click the  **Configure Transformation Scenario(s) (Ctrl (Meta on Mac OS) + Shift + C)** button on the **Transformation** toolbar, then click the **New** button and select **XML transformation with XSLT**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the  **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XML transformation with XSLT**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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 the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- **XSLT**;
- **Output**;
- **FO Processors**.

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: In case 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: In case 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 in case of *external XSLT processors*. In all other cases a non-empty XML URL value is mandatory.

- **XSL URL** - specifies the source XSL file that the transformation uses. 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 to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field;
- **Browse for local file** - opens a local file browser dialog box allowing to select a local file;
- **Browse for remote file** - opens an URL browser dialog box allowing to select a remote file;
- **Browse for archived file** - opens a zip archive browser dialog box allowing 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 in an editor panel the file with the path 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 "xml-stylesheet" declaration** - use the stylesheet declared with an `xml-stylesheet` 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 specified in the **XSL URL** field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction;
- **Transformer** - this combo box presents all the transformation engines available to Oxygen XML Editor plugin 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. In case no validation scenario is associated with an XSLT or XQuery document, the transformation engine is used in the validation process, if it provides validation support;
 - **Advanced options** - allows you to configure 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 available only in the **advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template which starts the XSLT transformation or the initial mode of 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, set with 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. In this case, you can copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line ;
- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XSLT elements used in the transformation;
- **Additional XSLT stylesheets** - opens *the dialog for adding XSLT stylesheets* which are applied on the result of the main stylesheet specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document;

XSLT Transformation Parameters

The parameters of the XSLT stylesheet used in the current transformation scenario.

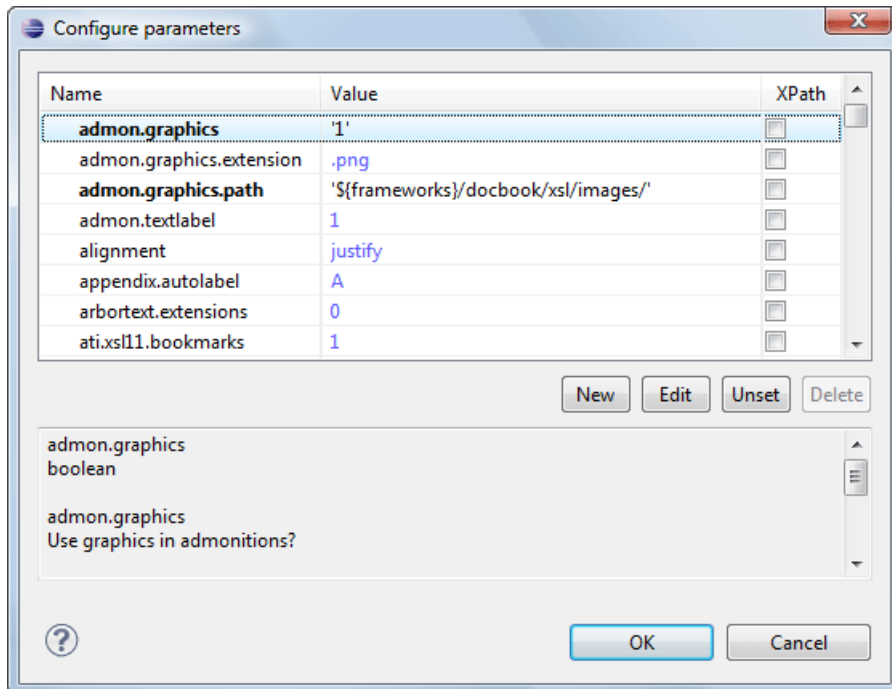


Figure 261: Configure parameters dialog

The table presents all the parameters of the XSLT stylesheet, all imported and included stylesheets and all *additional stylesheets* with their current values. The following font type and color conventions are used:

- blue font values are the defaults collected from the stylesheet;
- black font values and bold font names indicate edited parameters.

If a parameter value was not edited, then the table presents its default value. The bottom panel presents:

- 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.

For example setting the value of a parameter having a declared namespace like:

```
<xsl:param name="p:param" xmlns:p="namespace">default</xsl:param>
```

use the following expression in the **Name** column of the **Parameters** dialog:

```
{namespace}param
```

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 like:

```
doc('test.xml')//entry
//person[@atr='val']
```



Note:

1. The **doc** function solves the argument relative to the XSL stylesheet location. You can use full paths or editor variables like **\$(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 parameters:

- **New** - Adds a new parameter to the list.
- **Edit** - Edits the value of the selected parameter.
- **Unset** - Resets the selected parameter to its default value. Available only for parameters with set values.
- **Delete** - Removes the selected parameter from the list. It is enabled only for parameters added to the list with the **New** button.

The *editor variables* displayed at the bottom of the dialog can be used in the values of the parameters to make them independent of the location of the XSLT stylesheet or the XML document.

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 which stores the transformation result.
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables*.
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin opens the transformation result automatically, in a system application associated with the type of the result (HTML/XHTML, PDF, text) file.



Note: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes precedence over the default system application settings.



Note: Go to **Window > Preferences > General > Web Browser** to set the web browser that is used for displaying HTML/XHTML pages.

- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Editor plugin should open automatically at the end of the transformation the file specified in the **Save As** text field.
- **Other location** - When **Open in System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin should not open the file specified in the **Save As** text field, it should open the file specified in the text field of the **Other location** radio button. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variable*.
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on.
- **Show in results view as XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Editor plugin 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 time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **Show in results view as XML** - If this is checked Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents.
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path used to resolve image paths contained in the transformation result.

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 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*.

XSLT Stylesheet Parameters

The global parameters of the XSLT stylesheet used in the transformation scenario are configured from the dialog available from the **Parameters** button of the **Configure Transformation** dialog:

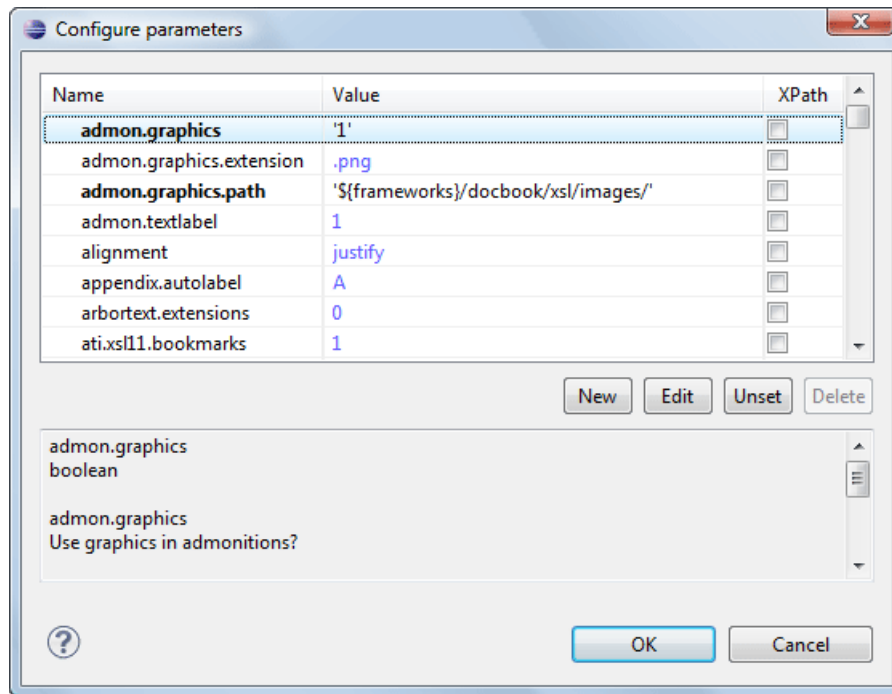


Figure 262: Configure parameters dialog

The table presents all the parameters of the XSLT stylesheet, all imported and included stylesheets and all *additional stylesheets* with their current values. The following font type and color conventions are used:

- blue font values are the defaults collected from the stylesheet;
- black font values and bold font names indicate edited parameters.

If a parameter value was not edited, then the table presents its default value. The bottom panel presents:

- 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

For example setting the value of a parameter having a declared namespace like:

```
<xsl:param name="p:param" xmlns:p="namespace">default</xsl:param>
```

use the following expression in the **Name** column of the **Parameters** dialog:

```
{namespace}param
```

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 like:

```
doc('test.xml')//entry
//person[@atr='val']
```



Note:

1. The **doc** function solves the argument relative to the XSL stylesheet location. You can use full paths or editor variables like **#{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 parameters:

- **New** - Adds a new parameter to the list.
- **Edit** - Edits the value of the selected parameter.
- **Unset** - Resets the selected parameter to its default value. Available only for parameters with set values.
- **Delete** - Removes the selected parameter from the list. It is enabled only for parameters added to the list with the **New** button.

The *editor variables* displayed at the bottom of the dialog can be used in the values of the parameters to make them independent of the location of the XSLT stylesheet or the XML document.

Additional XSLT Stylesheets

The list of additional XSLT stylesheets can be edited in the dialog opened by the button **Additional XSLT Stylesheets** from the **Configure Transformation** dialog.

- **Add** - Adds a stylesheet in the **Additional XSLT stylesheets** list using a file browser dialog. You can type an editor variable in the file name field of the browser dialog. 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.
- **Up** - Moves the selected stylesheet up in the list.
- **Down** - Moves the selected stylesheet down in the list.

The path specified in the URL text field can include *special Oxygen XML Editor plugin editor variables*.

XML Transformation with XQuery

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**;
- Click the **Configure Transformation Scenario(s) (Ctrl (Meta on Mac OS) + Shift + C)** button on the **Transformation** toolbar, then click the **New** button and select **XML transformation with XQuery**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XML transformation with XQuery**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or **Apply Transformation Scenario** 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 the **Storage** options:


- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- [The XQuery tab](#);
- [The FO Processor tab](#);
- [The Output tab](#).








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:** In case 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 uses. 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 to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field;
-  **Browse for local file** - opens a local file browser dialog box allowing to select a local file;
-  **Browse for remote file** - opens an URL browser dialog box allowing to select a remote file;
-  **Browse for archived file** - opens a zip archive browser dialog box allowing 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 in an editor panel the file with the path 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 combo box presents all the transformation engines available to Oxygen XML Editor plugin 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. In case no validation scenario is associated with an XSLT or XQuery document, the 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. If the XQuery/XSLT transformation engine is custom-defined you can not use this dialog to set parameters. Instead, copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT/XQuery engine to include the necessary parameters in the command line which starts the transformation process;
- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XQuery 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** - enable this option to avoid the long time necessary for fetching the full result. This option fetches only 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 which stores the transformation result;
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables*;
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin opens the transformation result automatically, in a system application associated with the type of the result (HTML/XHTML, PDF, text) file.



Note: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes precedence over the default system application settings.






Note: Go to **Window > Preferences > General > Web Browser** to set the web browser that is used for displaying HTML/XHTML pages.

- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Editor plugin should open automatically at the end of the transformation the file specified in the **Save As** text field;
- **Other location** - When **Open in System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin should not open the file specified in the **Save As** text field, it should open the file specified in the text field of the **Other location** radio button. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variable*
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on;
- **Show As XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Editor plugin 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 time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.
- **Show in results view as XML** - If this is checked Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents
- ;
- **Image URLs are relative to** - If **Show 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**;
- Click the  **Configure Transformation Scenario(s)(Ctrl (Meta on Mac OS) + Shift + T)** button on the **Transformation** toolbar, then click the **New** button and select **DITA OT Transformation**;

- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the  **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **DITA OT Transformation**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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. This dialog presents the list of possible outputs that the **DITA OT Transformation** is able to produce. Select the transformation type, click **OK** and move on to configuring the options in the **New Scenario** dialog. This dialog allows you to configure the options that control the transformation.


The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *Parameters*;
- *Filters*;
- *Advanced*;
- *Output*;
- *FO Processor*.



Note: To display the console during the transformation process, click  **Show console output** in the status bar.

The Parameters Tab

This dialog allows you to configure the parameters sent to the DITA-OT build file.

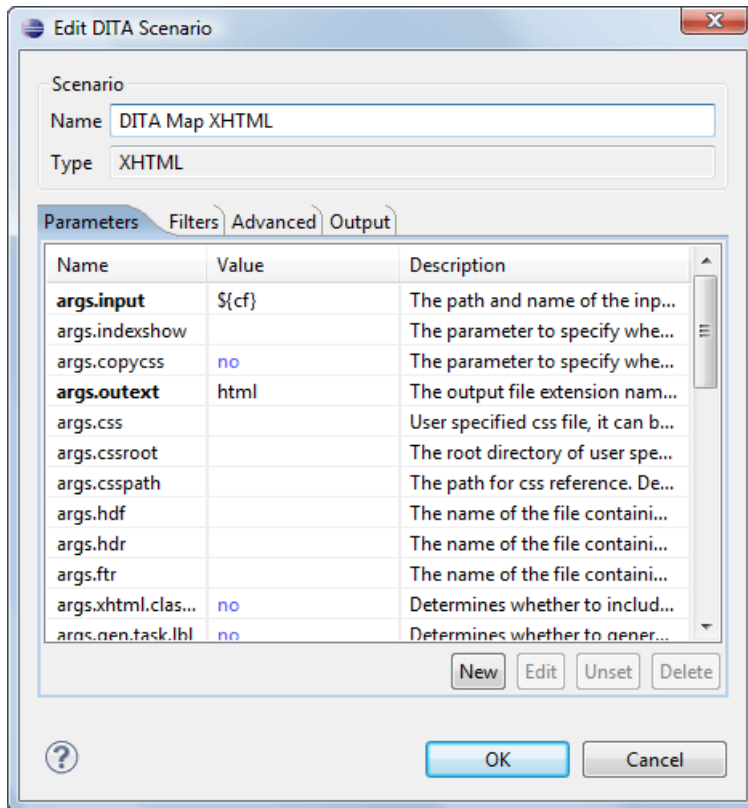


Figure 263: Edit DITA Ant transformation parameters

All the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (eg: XHTML) are listed along with their description. The values for some important parameters are already filled in. You can find more information about each parameter in the [DITA OT Documentation](#). You can also add additional parameters to the list.

Using the toolbar buttons you can add, edit or remove a parameter.

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 variables selector to simplify setting a file path as value to a parameter.

The value of a parameter can be entered at runtime if a value `ask('user-message', param-type, 'default-value' ?)` is used as value of parameter in the Configure parameters dialog.

Examples:

- `${ask('message')}` - Only the message displayed for the user is specified.
- `${ask('message', generic, 'default')}` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
- `${ask('message', password)}` - 'message' is displayed, the characters typed are masked with a circle symbol.
- `${ask('message', password, 'default')}` - same as before, the default value is 'default'.
- `${ask('message', url)}` - 'message' is displayed, the parameter type is URL.
- `${ask('message', url, 'default')}` - same as before, the default value is 'default'.

The Filters Tab

In the scenario **Filters** tab you can add filters to remove certain content elements from the generated output.

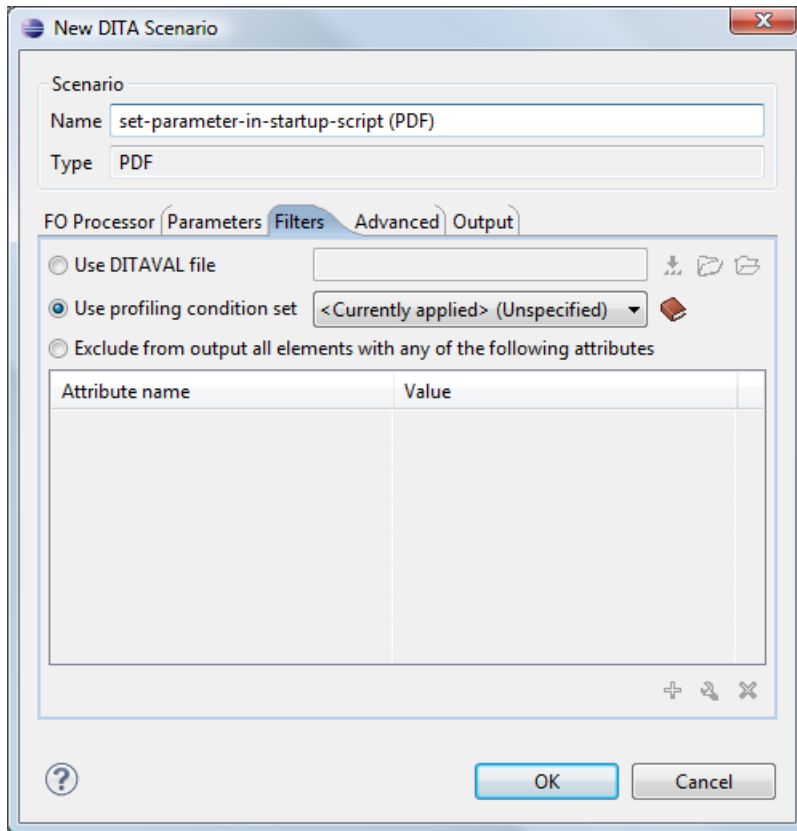


Figure 264: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - if you already have a DITAVAL file associated with the transformed map you can specify the path to it and it will be used when filtering content. You can find out more about constructing a DITAVAL file in the [DITA OT Documentation](#) topic;
- **Use profiling condition set** - sets the *profiling condition set* that applies to the document you transform;
- **Exclude from output all elements with any of the following attributes** - you can configure a simple list of attribute (name, value) pairs which when present on an element in the input will remove it from output.

The *Advanced* Tab

In the **Advanced** tab, you can specify advanced options for the transformation.

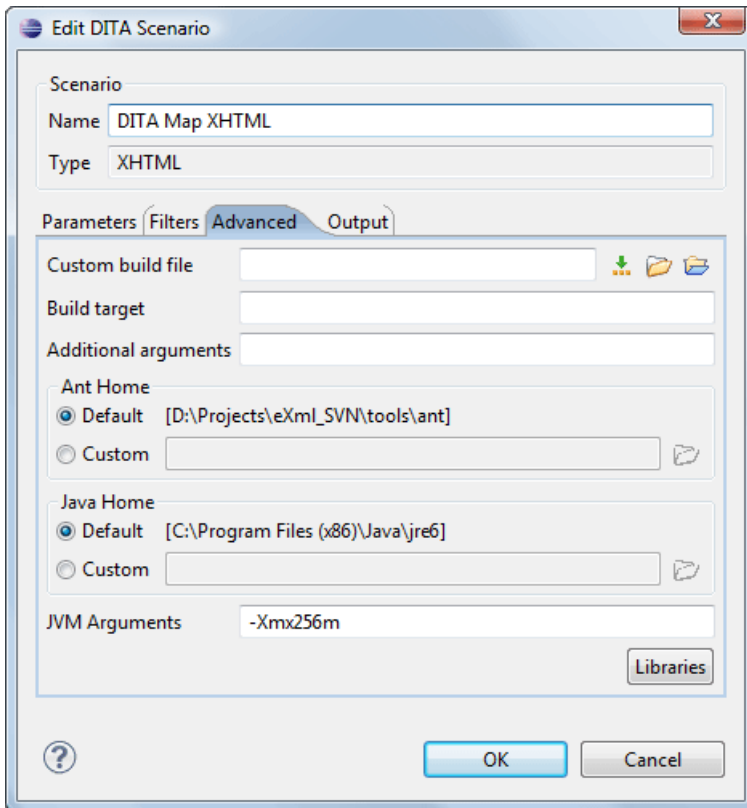


Figure 265: Advanced settings tab

You have several parameters that you can specify here:

- **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` directory configured in the **Parameters** tab is used.
- **Build target** - You can specify a build target to the build file. By default no target is necessary and the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation like `-verbose`.
- **Ant Home** - You can specify a custom ANT installation to run the DITA Map transformation. By default it is the ANT installation bundled with .
- **Java Home** - You can specify a custom Java Virtual Machine to run the ANT transformation. By default it is the Java Virtual Machine used by .
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to `-Xmx384m` which means the transformation process is allowed to use 384 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (384 MB) to a higher value, like 512 MB. This way, you can avoid the Out of Memory error messages (**OutOfMemoryError**) received from the ANT process.



Note: If you are publishing DITA to PDF and still experience problems, you should also increase the amount of memory allocated to the FO transformer. To do this, go to the **Advanced** tab and increase the value of the **Java Arguments** parameter.

- **Libraries** - adds by default as high priority libraries which are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can specify all the additional libraries (jar files or additional class paths) which are used by the ANT transformer. You can also decide to control all libraries added to the classpath.

The Output Tab

In the **Output** tab, you can configure options related to the place where the output is generated.

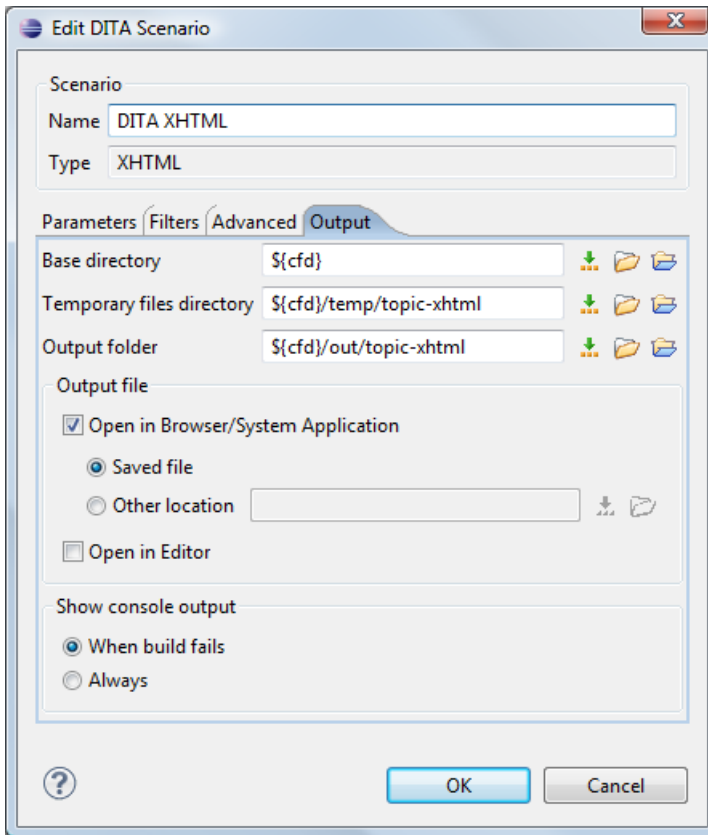



Figure 266: Output settings tab

You have several parameters that you can specify here:

- **Base directory** - all the relative paths which appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located;
- **Temporary files directory** - this directory is used to store pre-processed temporary files until the final output is obtained;
- **Output folder** - the folder where the final output content is copied;
- **Output file options** - the transformation output can then be opened in a browser or even in the editor, if specified;
- **Show console output** - specifies whether the console is always displayed or only when the build fails.

 **Note:** If the DITA Map or topic is opened from a remote location or from a ZIP file, the scenario must specify absolute output, temporary and base file paths.

The FO Processor Tab

This tab allows you to set an FO Processor, when you choose to generate PDF output.

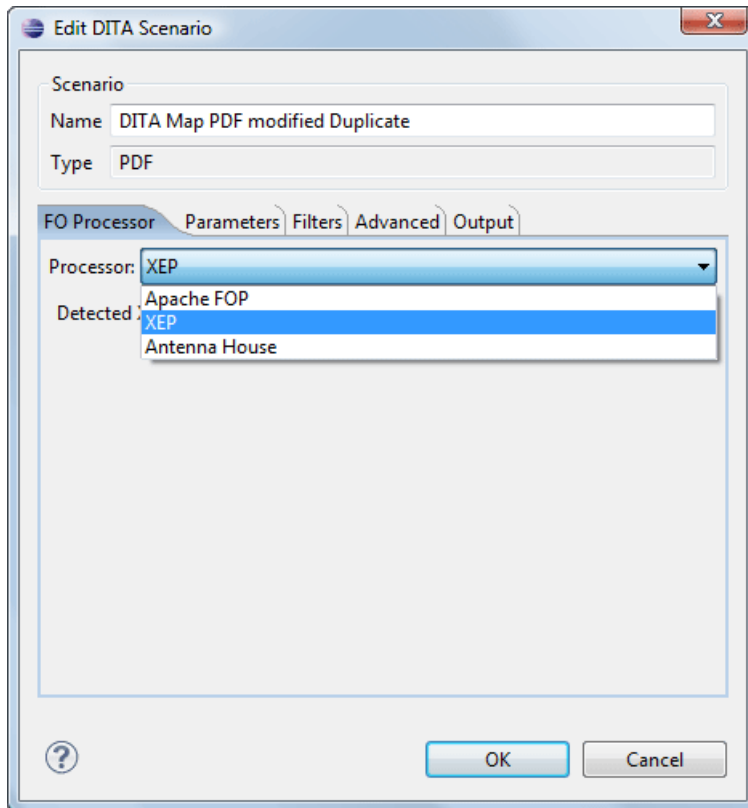


Figure 267: FO Processor configuration tab

You can choose between:

- **Apache FOP** - Default setting. This processor comes bundled with .
- **XEP** - The *RenderX* XEP processor.

If you select **XEP** in the combo and XEP was already installed in you can see the detected installation path appear under the combo box.

XEP is considered as 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 (eg: `xep.bat` on Windows);
- XEP was installed in the `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 was already installed on your computer and you select **Antenna House** in the combo box, in you can see the detected installation path appear under the combo.

Antenna House is considered as 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.

ANT Transformation



An ANT transformation scenario is usually associated with an Ant build script. Oxygen XML Editor plugin 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**, select  **Transformation Scenarios**, click **New**, and select **ANT**;
- Click the  **Configure Transformation Scenario(s) (Ctrl (Meta on Mac OS) + Shift + C)** button on the **Transformation** toolbar, then click the **New** button and select **ANT**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the  **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **ANT**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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 the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:



- *the Options tab;*
- *the Parameters tab;*
- *the Output tab.*



Note: To display the console during the transformation process, click  **Show console output** in the status bar.

The Options Tab

The **Options** tab contains the following options:

- **Working directory** - path of the current directory of the Ant external process. An *editor variable* can be inserted in this text box using the small green arrow button (
- **Build file** - ant script file that is the input of the Ant external process. An *editor variable* can be inserted in this text box using the small green arrow button (
- **Build target** - optionally a build target from the Ant script file can be specified. If no target is specified, the Ant target that is specified as default in the Ant script file is executed;
- **Additional arguments** - additional command-line arguments to be passed to the Ant transformation (for example -verbose);
- **Ant Home** - path to the Ant installation to run the transformation. By default it is the Ant installation version 1.8.2 that is bundled with Oxygen XML Editor plugin. A custom Ant installation can also be set;
- **Java Home** - the path to the Java Virtual Machine that runs the Ant transformation. By default it is the Java Virtual Machine that is bundled with Oxygen XML Editor plugin. A custom Java virtual machine can also be set;
- **JVM Arguments** - this parameter allows you to set specific parameters to the Java Virtual Machine used by Ant. By default it is set to -Xmx384m which means the transformation process is allowed to use 384 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (384 MB) to a higher value, like 512 MB. In this way, you can avoid running out of memory (**OutOfMemoryError**) when running an Ant process;


- **Libraries** - this button allows adding to the classpath of the Ant process any external libraries that are not bundled with Ant (that is they are not built-in Ant libraries).

The Parameters Tab

On the **Parameters** tab you can use the **New**, **Edit**, and **Delete** buttons to set the parameters accessible as Ant properties in the Ant build script.




The Output Tab

The following details can be configured in the **Output** tab:



- the file to open automatically when the transformation is finished in the **Open** text box, usually the output file of the Ant process; an *editor variable* can be inserted in this text box using the small green arrow button ();
- if 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 files (for example the *Acrobat Reader* application for *.pdf* files);
- if the file specified in the **Open** text box is opened in Oxygen XML Editor plugin; for example if it is an *.xml* file it is opened automatically in *the XML editor panel*, if it is a *.zip* file or an *.epub* file it is opened in *the Archive Browser view*, etc.;

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**;
- Click the  **Configure Transformation Scenario(s) (Ctrl (Meta on Mac OS) + Shift + C)** button on the **Transformation** toolbar, then click the **New** button and select **XSLT transformation**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the  **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XSLT transformation**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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 the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- **XSLT**;
- **FO Processor**;
- **Output**;

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - specifies the source XML file that the transformation uses. 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: In case 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: In case 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 in case of *external XSLT processors*. In all other cases a non-empty XML URL value is mandatory.

- **XSL URL** - specifies the source XSL. 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 to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field;
- **Browse for local file** - opens a local file browser dialog box allowing to select a local file;
- **Browse for remote file** - opens an URL browser dialog box allowing to select a remote file;
- **Browse for archived file** - opens a zip archive browser dialog box allowing 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 in an editor panel the file with the path 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 "xml-stylesheet" declaration** - use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default this checkbox is not selected and the transformation uses the XSLT stylesheet specified in the **XSL URL** field. If it is checked, the scenario is applied to the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction;
- **Transformer** - this combo box presents all the transformation engines available to Oxygen XML Editor plugin 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. In case no validation scenario is associated with an XSLT or XQuery document, the transformation engine is used in the validation process, if it provides validation support;
- **Advanced options** - allows you to configure 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 available only in the **advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template which starts the XSLT transformation or the initial mode of 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, set with 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. In this case, you can copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line ;
- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XSLT elements used in the transformation;
- **Additional XSLT stylesheets** - opens *the dialog for adding XSLT stylesheets* which are applied on the result of the main stylesheet 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 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 which stores the transformation result;
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables*;
- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Editor plugin should open automatically at the end of the transformation the file specified in the **Save As** text field;
- **Other location** - When **Open in System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin should not open the file specified in the **Save As** text field, it should open the file specified in the text field of the **Other location** radio button. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variable*;
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on;
- **Show in results view as XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Editor plugin 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 time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.
- **Show in results view as XML** - If this is checked Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents;
- ;
- **Image URLs are relative to** - If **Show 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**;
- Click the  **Configure Transformation Scenario(s) (Ctrl (Meta on Mac OS) + Shift + C)** button on the **Transformation** toolbar, then click the **New** button and select **XProc transformation**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the  **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XProc transformation**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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 the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *the XProc tab;*
- *the Inputs tab;*
- *the Parameters tab;*
- *the Outputs tab;*
- *the Options tab.*

The XProc Tab

The **XProc** tab contains the following options:

- **XProc URL** - specifies the URL of the XProc script;
- **Processor** - specifies the XProc engine. You can select the built-in *Calabash* engine or a custom engine *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. To add, modify, and delete ports from this list, use the **New**, **Edit**, and **Delete** buttons. Each input port has an assigned name in the XProc script. The XProc engine reads data from the URLs specified in the **URLs** list. The *built-in editor variables* and the *custom editor variables* can be used to specify these URLs.

The Parameters Tab

The **Parameters** tab presents a list with the parameters collected from the XProc script. To add new parameters click the **New** button.

Each port where the output of the XProc transformation is sent is associated with an URL on the **Outputs** tab of the dialog. The *built-in editor variables* and the *custom editor variables* can be used for specifying this URL.

The Outputs Tab

The **Outputs** tab displays a list of output ports collected from the XProc script. To define additional output ports click the **New** button .

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 of the document(s) that correspond to the selected output port on the right side.

If the **Open results in editor** option is selected, the XProc transformation result is opened automatically in an editor panel. By selecting the **Open in System Application** option, you can specify a file that is opened at the end of the XProc transformation in the system application associated with that file type.

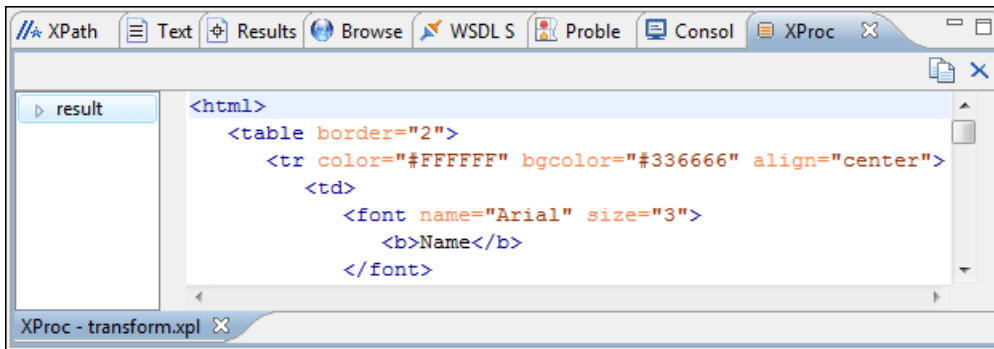


Figure 268: XProc Transformation results view

The Options Tab

The **Options** displays a list with the options collected from the XProc script. To define new options click the **New** button.

The options collected from the script are rendered plain. Edited options as well as new ones are rendered bold.

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-install-folder]/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 plugin.

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](#). In order to create an XProc integration project follow the next steps:

1. Take the `oxygen.jar` from `[Oxygen-install-folder]/lib` and put it in 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 have the following meanings:
 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 this integration.
 5. `engineVersion` - The version of the integrated engine.
 6. `vendor` - The name of the vendor / implementor.
 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-install-folder]/lib/xproc`.



- Put there the `engine.xml`, and all the libraries necessary to run the new integration.

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**;
- Click the  **Configure Transformation Scenario(s)** (**Ctrl (Meta on Mac OS) + Shift + C**) button on the **Transformation** toolbar, then click the **New** button and select **XQuery transformation**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the  **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XQuery transformation**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or  **Apply Transformation Scenario** 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 the **Storage** options:

- Global Options** - the scenario is saved in the global options stored in the user home directory;
- Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- the XQuery tab;*
- the FO Processor tab;*
- the Output tab.*

The XQuery Tab

The **XQuery** tab contains the following options:








- XML URL** - specifies the source XML file that the transformation uses. 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: In case 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. 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 to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field;
-  **Browse for local file** - opens a local file browser dialog box allowing to select a local file;
-  **Browse for remote file** - opens an URL browser dialog box allowing to select a remote file;
-  **Browse for archived file** - opens a zip archive browser dialog box allowing 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 in an editor panel the file with the path 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 combo box presents all the transformation engines available to Oxygen XML Editor plugin 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. In case no validation scenario is associated with an XSLT or XQuery document, the 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. If the XQuery/XSLT transformation engine is custom-defined you can not use this dialog to set parameters. Instead, copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT/XQuery engine to include the necessary parameters in the command line which starts the transformation process
- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XQuery 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 tab:

- **Prompt for file** - At the end of the transformation a file browser dialog is displayed for specifying the path and name of the file which stores the transformation result.
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables*.
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin opens the transformation result automatically, in a system application associated with the type of the result (HTML/XHTML, PDF, text) file.



Note: If you already set the **Default Internet browser** option in the **Global** preferences page, it takes precedence over the default system application settings.



Note: Go to **Window > Preferences > General > Web Browser** to set the web browser that is used for displaying HTML/XHTML pages.

- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Editor plugin should open automatically at the end of the transformation the file specified in the **Save As** text field.
- **Other location** - When **Open in System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin should not open the file specified in the **Save As** text field, it should open the file specified in the text field of the **Other location** radio button. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variable*.
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on.
- **Show As XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Editor plugin 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 time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **Show As XML** - If this is checked Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents.
- **Image URLs are relative to** - If **Show 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**;
- Click the **Configure Transformation Scenario(s)** (**Ctrl (Meta on Mac OS) + Shift + C**) button on the **Transformation** toolbar, then click the **New** button and select **SQL transformation**;
- Select **Ctrl (Meta on Mac OS) + Shift + T** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **SQL transformation**.



Note: In case a scenario is already associated with the edited document, selecting **Ctrl (Meta on Mac OS) + Shift + T** or **Apply Transformation Scenario** 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 transformation scenario;
- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined;
- **SQL URL** - specifies the URL of the SQL script. This field also accepts *editor variables*.
- **Connection** - opens the *data source preferences page*;
- **Parameters** - allows you to configure the parameters of the transformation.

XSLT/XQuery Extensions

The **Libraries** dialog is used to specify the jars and classes containing extension functions called from the XSLT/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 the dialog. For changing the order of the items the user must select the item that must be moved to other position in the list and press the up and down buttons.

The Configure Transformation Scenario(s) Dialog

You can use this dialog to manage both the *built-in transformation scenarios* and the ones you create.

To open it, either click the **Configure Transformation Scenario(s)** button on the application toolbar, or go to **XML > Configure transformation scenario**. (**Alt+Shift+T C**) (**Cmd+Alt+T C on Mac OS**).

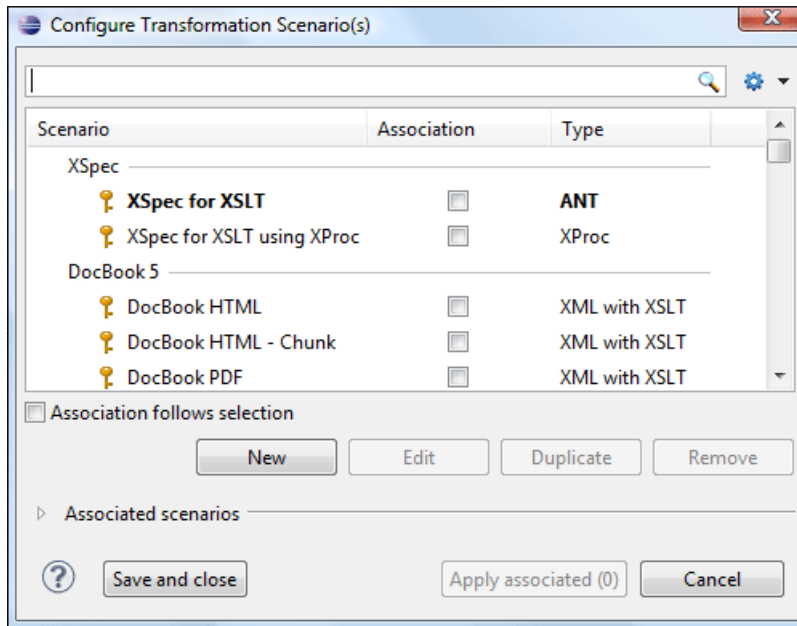





Figure 269: Configure Transformation Scenario(s) Dialog

The top section of the dialog contains a filter that allows you to search through the scenarios list. Using the **Settings** button you can 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 display in the **Transformation scenarios** view only the scenarios which Oxygen XML Editor plugin can execute for the current document type;
- **Show associated scenarios** - select this option to display only the scenarios associated with the document you are editing;
-  **Import scenarios** - use this option to open the **Import scenarios** dialog. The **Import scenarios** dialog allows you to select which scenario of the `scenarios` files you want to import. In case one of the scenarios you import is identical with an existing scenario, Oxygen XML Editor plugin ignores the scenario you are importing. In case a conflict appears (an imported scenario has the same name with 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 plugin 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 plugin creates a `scenarios` file which contains the scenarios you export.

The middle section of the dialog presents the scenarios that you can apply to the document you are editing. 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 document you are editing;
- **Scenario** - this column presents the names of the transformation scenarios;
- **Type** - specifies the type of the transformation scenario. For further details about the different types of transformation scenarios available in Oxygen XML Editor plugin go to [Defining a New Transformation Scenario](#);
- **Storage** - specifies whether a transformation scenario is defined at project level.

Left click the header of each column to sort its items. The contextual menu of each header allows you to show or hide the **Type** and **Storage** columns, group the columns by **Association**, **Type**, and **Storage**, ungroup all of them, or reset the layout.

The bottom section of the dialog contains the following actions:

- **Association follows selection** - enable this check-box to associate selected transformation scenarios automatically with the document you are editing. This option also works for multiple selection;



Note: When this option is enabled, the **Association** column is no longer presented.

- **New** - this button allows you to create a new transformation scenario *depending on its type*;
- **Edit** - this button opens the **Edit scenario** dialog which allows you to configure the options of the transformations scenario you are editing;



Note: In case you try to edit a transformation scenario associated with a document type, Oxygen XML Editor plugin displays a warning message to inform you that this is not possible. You have the option to create a *duplicated transformation scenario* and edit this one instead.

- **Duplicate** - use this button to create a *duplicated transformation scenario*;
- **Remove** - use this button to remove transformation scenarios.



Note: Removing scenarios associated with a document type is not permitted.

The **Edit**, **Duplicate**, and **Remove** actions are also available in the contextual menu of the transformation scenarios listed in the middle section of the **Configure Transformation Scenario(s)**. This contextual menu also contains the *Import scenarios* and *Export selected scenarios* actions and allows you to change the storage of a transformation scenario.

Duplicating a Transformation Scenario

Use the following procedure to duplicate a transformation scenario.

1. Go to menu **XML > Configure Transformation Scenario (Alt+Shift+T C) ((Meta+Alt+T C on Mac OS))** to open the **Configure Transformation** dialog.
2. Click the **Duplicate Scenario** button of the dialog to create a copy of the current scenario.
3. Click the **Name** field and type a new name.
 - a) You can choose to save the scenarios at project level by setting the **Project Scenarios** setting.
4. Click **OK** or **Transform Now** to save the scenario.


Editing a Transformation Scenario

Editing a transformation scenario is useful in case you run an existing one and you need to configure some of its parameters.


Oxygen XML Editor plugin allows you to configure exiting transformation scenarios either from the **Transformation Scenarios** view, or from the **Configure Transformation Scenario(s)**, and **Transform with** dialog boxes.

Use one of the following methods to configure a transformation scenario:

- go to the **Transformation Scenarios** view, select a transformation scenario and click the **Edit** button on the toolbar of the view;
- click the **Configure Transformation Scenario(s)** button on the **Transformation** toolbar, select a transformation scenario from the **Configure Transformation Scenario(s)** dialog box and click the **Edit** button;

- click the  **Apply Transformation Scenario** button on the **Transformation** toolbar, select a transformation scenario from the **Transform With** dialog box and click the **Edit** button.






Note: In case a transformation scenario is associated with the document you are editing, selecting the  **Apply Transformation Scenario** button starts the transformation process automatically.

You can edit transformation scenarios that are defined at project level only. To edit a transformation scenario that is associated with a document type, duplicate it and edit the duplicated scenario.

Batch Transformation

A transform action can be applied on a batch of files *from the Project view's contextual menu* without having to open the files involved in the transformation:

-  **Apply Transformation Scenario(s)** - applies the transformation scenario associated to each of the selected files. If the currently processed file does not have an associated transformation scenario then a warning is displayed in the **Warnings** view;
-  **Configure Transformation Scenario(s)...** - opens a dialog box that allows you to create, manage and set default transformation scenarios;
-  **Transform with...** - Allows you to select one transformation scenario to be applied to each one of the currently selected files.


xsl:message output information is collected and displayed in the *Results View*. All entries in the Results View point to the location of the code that triggered them.




Note: When you trigger a batch transformation, the output messages from the previous one are deleted.

Built-in Transformation Scenarios

Oxygen XML Editor plugin comes with preconfigured built-in transformation scenarios used for usual transformations.

To obtain the output simply one of the built-in scenarios with the currently edited document and click the  **Apply Transformation Scenario(s)** button.

You can use the  **Apply Transformation Scenario(s)** button even if the document you are editing is not associated with a transformation scenario.

In case the document contains an `xml-stylesheet` processing instruction referring to an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Editor plugin 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. The **Use xml-stylesheet declaration** check-box of the scenario is selected by default. Saxon is used as transformation engine and no FO processing is performed. The result of the transformation is store 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*: `${cfd}` and `${cfn}`.

Transformation Scenarios View

You are able to manage the transformation scenarios easily, using the **Transformation Scenarios** view. To open this view, go to **Window > Show View > Transformation Scenarios**.

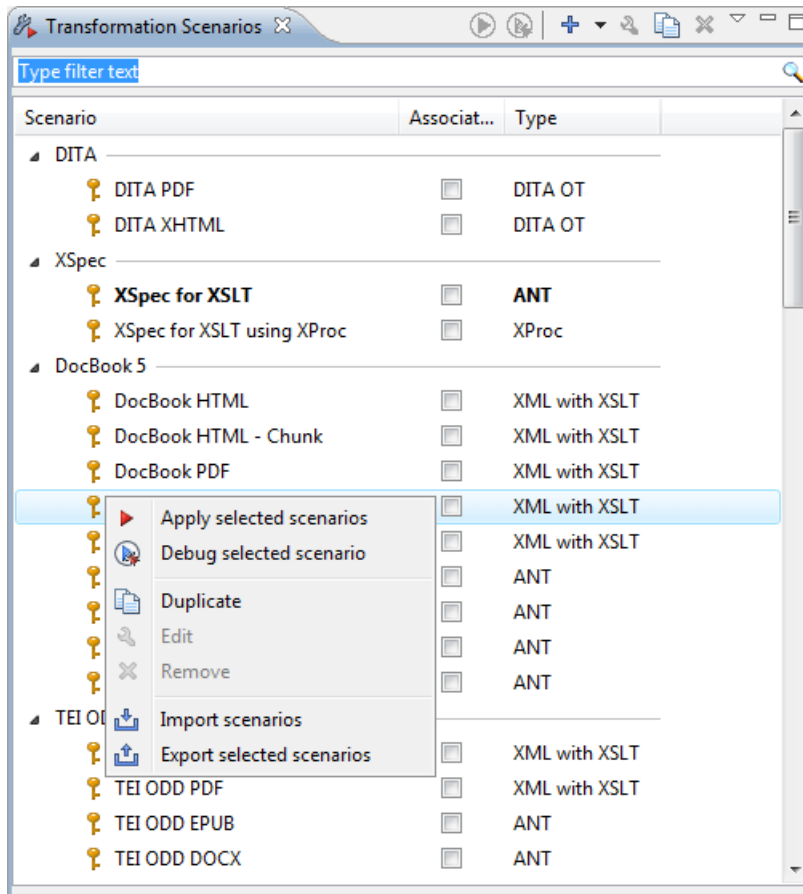


Figure 270: The Scenarios view

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 input, the 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 the same as an existing one but needs some changes;
 - **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 on the **Delete** key;
 - **Import scenarios** - use this option to open the **Import scenarios** dialog. The **Import scenarios** dialog allows you to select which scenario of the `scenarios` file you want to import. In case one of the scenarios you import is identical with an existing scenario, Oxygen XML Editor plugin ignores the scenario you are importing. In case a conflict appears (an imported scenario has the same name with 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 plugin 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 plugin creates a `scenarios` file which contains the scenarios you export;

Apart from the options available in the contextual menu, the **Transformation Scenarios** view toolbar contains a





New drop-down button. The menu of this drop down button contains a list of the scenarios you can create. Oxygen XML Editor plugin analyses which scenario is appropriate and displays it as the first item in the list of scenarios, separated by the rest through a horizontal line.

The  **Settings** drop-down menu of the **Transformation Scenarios** view 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 display in the **Transformation scenarios** view only the scenarios which Oxygen XML Editor plugin can execute for the current document type;
- **Show associated scenarios** - select this option to display only the scenarios associated with the document you are editing;
- **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** - use this option to open the **Import scenarios** dialog. The **Import scenarios** dialog allows you to select which scenario of the `scenarios` file you want to import. In case one of the scenarios you import is identical with an existing scenario, Oxygen XML Editor plugin ignores the scenario you are importing. In case a conflict appears (an imported scenario has the same name with 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 plugin 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 plugin creates a `scenarios` file which contains the scenarios you export;
- **Show Type** - use this option to display the transformation type of each scenario in the **Transformation Scenarios** view;
- **Show Storage** - use this option to display the storage location of the scenarios in the **Transformation Scenarios** view;
- **Group by Association** - select this option to group the scenarios in the **Transformation Scenarios** view depending on whether they are associated or not with the document you are editing;
- **Group by Type** - select this option to group the scenarios in the **Transformation Scenarios** view by their type;
- **Group by Storage** - select this option to group the scenarios in the **Transformation Scenarios** view by their storage location;
-  **Ungroup all** - select this option to ungroup all the scenarios of the **Transformation Scenarios** view;
- **Reset Layout** - select this option to restore the default settings of the layout of the **Transformation Scenarios** view.

Oxygen XML Editor plugin 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 plugin 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 Type** - select this option to group the scenarios in the **Transformation Scenarios** view by their type;
- **Group by Storage** - select this option to group the scenarios in the **Transformation Scenarios** view by their storage location;

Using the Oxygen WebHelp Plugin

The Oxygen WebHelp Plugin allows you to transform DITA and Docbook documents outside of Oxygen XML Editor plugin, from a command line. The WebHelp output files created with the Oxygen WebHelp plugin are the same with the output files that Oxygen XML Editor plugin produces when you run DITA or DocBook to WebHelp transformation scenarios.

Oxygen WebHelp Plugin for DITA

To transform DITA documents using the Oxygen 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** for desktop;
- **webhelp-feedback** - the transformation that produces **webhelp** for desktop with the addition of a feedback system;
- **webhelp-mobile** - the transformations that produces **webhelp** for mobile devices.

Integrating the Oxygen WebHelp Plugin with the DITA Open Toolkit

The requirements of the Oxygen WebHelp plugin for the DITA Open Toolkit are:

- Java Virtual Machine 1.6 or later;
- DITA Open toolkit 1.6.x or 1.7.x (Full Easy Install);
- Saxon 9.1.0.8.

To integrate the Oxygen 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 or 1.7.x (Full Easy Install).
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. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.
6. 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 WebHelp Plugin

To register the Oxygen WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Create a `.txt` file named `license` in the `[DITA-OT-install-dir]/plugins/com.oxygenxml.webhelp` directory. The `license.txt` file is created.
2. In this file, copy your Oxygen Scripting license key which you purchased for your Oxygen Webhelp plugin. The WebHelp transformation process reads the Oxygen 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 WebHelp Plugin

To run a DITA to WebHelp (**webhelp**, **webhelp-feedback**, **webhelp-mobile**) transformation using the Oxygen 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 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 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=/path/to/directory/of/header-file.html
```

where `/path/to/directory/of/header-file.html` is the location of the directory that contains the header file.

Database Configuration for DITA WebHelp with Feedback

In case you run the **webhelp-feedback** transformation, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `instalation.html` file, located at `[DITA-map-base-dir]/out/[transtype]/oxygen-webhelp/resources`. The `instalation.html` file is created by the transformation process.

Oxygen WebHelp Plugin for DocBook

To transform DocBook documents using the Oxygen 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** for desktop;
- **webhelp-feedback** - the transformation that produces **webhelp** for desktop with the addition of a feedback system;
- **webhelp-mobile** - the transformations that produces **webhelp** for mobile devices.

Integrating the Oxygen 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 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 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 [saxon9-1-0-8j.zip](#) on your computer.

Registering the Oxygen WebHelp Plugin

To register the Oxygen 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 Oxygen Scripting license key, which you purchased for your Oxygen WebHelp plugin.

The WebHelp transformation process reads the Oxygen 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 DocBook Transformation Using the WebHelp Plugin

To run a DocBook to WebHelp (**webhelp**, **webhelp-feedback**, **webhelp-mobile**) transformation using the Oxygen 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 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 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/stylesheets.css
```

Database Configuration for DocBook WebHelp with Feedback

In case you run the **webhelp-feedback** transformation, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `instalation.html` file, located at `[OUTPUT_DIR]/oxygen-webhelp/resources/instalation.html`. The `instalation.html` file is created by the transformation process.

XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Editor plugin.

Supported XSLT Processors

Oxygen XML Editor plugin comes with 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, which 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 9.5.0.1 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 were present in Saxon-PE.
- **Saxon 9.5.0.1 Enterprise Edition (EE)** - *Saxon EE* is the schema-aware edition of Saxon and it is one of the built-in processors of Oxygen XML Editor plugin. 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 plugin implements a Saxon framework that allows you to create Saxon configuration files. Two templates are available: **Saxon collection catalog** and **Saxon configuration**. Both these templates support content completion, element annotation and attribute annotation.

Besides the above list Oxygen XML Editor plugin supports the following processors:

- **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 expressions extensions. The `libxml2` version included in Oxygen XML Editor plugin is 2.7.6 and the `libxslt` version is 1.1.26

Oxygen XML Editor plugin uses `Libxslt` through its command line tool (`Xsltproc`). The XSLT processor is included into the distribution kit of the stand-alone version for Windows and Mac OS X. Because there are differences between different 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.

If you do not have the `Libxslt` library already installed, you should copy the following files from Oxygen XML Editor plugin stand-alone installation directory to the root of the `com.oxygenxml.editor_15.2` plugin:

- on Windows: `xsltproc.exe`, `zlib1.dll`, `libxslt.dll`, `libxml2.dll`, `libexslt.dll`, `iconv.dll`
- on Linux: `xsltproc`, `libexslt.so.0`, `libxslt.so.1`, `libxml2.so.2`
- on Mac OS X: `xsltproc.mac`, `libexslt`, `libxslt`, `libxml`

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 by LIBXML if Oxygen XML Editor plugin is installed in the default location on Windows (C:\Program Files) because the built-in XML catalog files are stored in the `frameworks` subdirectory of the installation directory which in this case contains at least a space character.

- **MSXML 3.0/4.0** - *MSXML 3.0/4.0* is available only on Windows 2000, Windows NT and Windows XP platforms. It can be used for [transformation](#) and [validation of XSLT stylesheets](#).

Oxygen XML Editor plugin uses the Microsoft XML parser through its command line tool `msxsl.exe`.

Because `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 NT4, Windows 2000 and Windows XP platforms. It can be used for *transformation* and *validation of XSLT stylesheets*.

Oxygen XML Editor plugin 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 plugin 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](#)

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 plugin 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 plugin 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 referred 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.

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 9.5.0.1 - <http://www.saxonica.com/documentation/extensibility/intro.xml>

To set an XSLT processor extension (a directory or a jar file), use [the *Extensions* button](#) of the scenario edit dialog. The old way of setting an extension (using the parameter `-Dcom.oxygenxml.additional.classpath`) was deprecated and 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 plugin.

The Built-in XSL-FO Processor

The Oxygen XML Editor plugin 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 plugin 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 `mliwrapper_jai.jar` for *JAI*. For the *JAI* package you can include the directory containing the native libraries (`mli_jai.dll` and `mli_jai_mmx.dll` on Windows) in the *PATH* system variable.

The Mac 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](#).

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">
      <fonts>
        <auto-detect/>
      </font>
    </renderer>
  </renderers>
</fop>
```

```
</renderers>
</fop>
```

- b) Set the FOP configuration file in **Preferences**.

Go to menu **Options > Preferences > XML > XSLT/FO/XQuery > FO Processors** and enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** 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:
 - `${frameworks}/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml` - the `font-face` element included in each element `physical-font` having the attribute `char-set="default"` must contain the name of the font (**Arial Unicode MS** in our example)
 - `${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">
  .
  <fonts>
    <auto-detect/>
  </fonts>
  .
</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 plugin install directory.
- Create the following script file in the Oxygen XML Editor plugin installation directory.

For Mac 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.fonts.apps.TTFReader"
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.TTFReader
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 plugin 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 Mac 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 Mac 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>
      <font>
        <font metrics-url="Arialuni.xml" kerning="yes"
          embed-url="file:/Library/Fonts/Arialuni.ttf">
          <font-triplet name="Arialuni" style="normal"
            weight="normal"/>
        </font>
      </font>
    </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>
    <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="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) Set the FOP configuration file in **Preferences**.

Go to menu **Options > Preferences > XML > XSLT/FO/XQuery > FO Processors** and enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** 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:

- `${frameworks}/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml` - the `font-face` element included in each element `physical-font` having the attribute `char-set="default"` must contain the name of the font (*Arialuni* in our example)
- `${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 install folder]/lib/fop` directory.

Hyphenation

To add support for hyphenation:

1. download the pre-compiled JAR from [OFFO](#) ;
2. place the JAR in `[oxygen install folder]/lib/fop`;
3. restart the Oxygen XML Editor plugin.

Chapter 11

Querying Documents

Topics:

- [Running XPath Expressions](#)
- [Working with XQuery](#)

This chapter shows how to query XML documents in Oxygen XML Editor plugin with XPath expressions and the XQuery language.

Running XPath Expressions

This section covers the views, toolbars, and dialogs in Oxygen XML Editor plugin, 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>.

The XPath Dialog

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](#).

Oxygen XML Editor plugin provides a specialized **XPath** dialog to let you query XML documents fast and easy using XPath expressions (XPath 1.0, XPath 2.0, XPath 2.0 SA, XPath 3.0, XPath 3.0 SA). To open this dialog, either click the **//*** **XPath** toolbar button, or go to **XML > XPath**. You can use the **XPath** dialog to execute both XPath 2.0 basic and XPath 2.0 schema aware expressions. XPath 2.0 schema aware takes into account the Saxon EE [XML Schema version](#) option.

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 plugin 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 plugin can return a compilation error.

Syntax highlight is available in the XPath dialog and allows you to identify the components of an XPath expression. To customize the colors of the components of an XPath expression, go to **Window > Preferences > Oxygen XML Editor > Editor > Syntax Highlight**.

The **Content Completion Assistant** is also available in the **XPath** dialog. It offers context-dependent proposals. The set of XPath functions proposed by the **Content Completion Assistant** also depends on the XPath version. Select the XPath version from the **XPath** field.

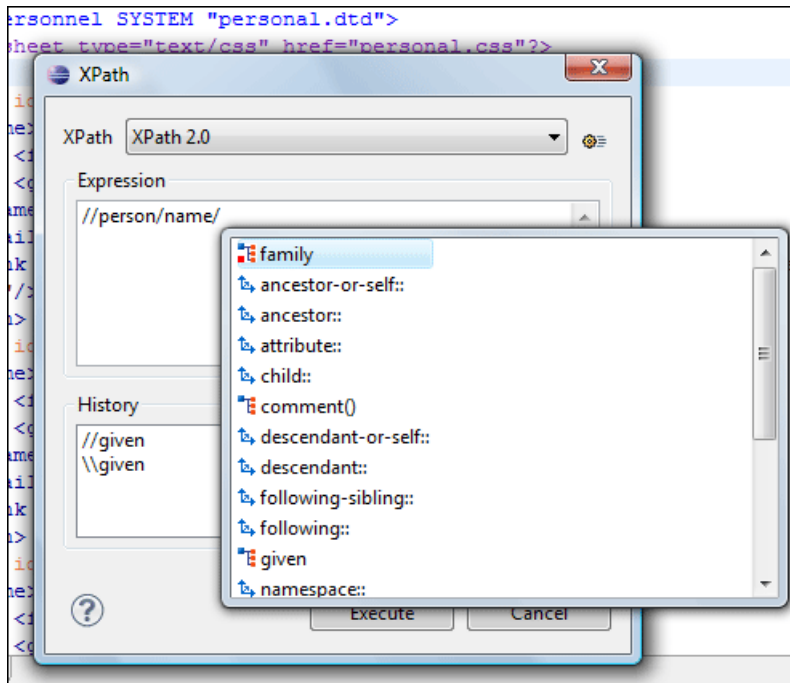


Figure 271: Content Completion in the XPath dialog

When Oxygen XML Editor plugin evaluates an XPath expression, it uses [XML catalogs](#) to resolve the location of documents referred in the expression. To configure the XML catalogs, go to **Window > Preferences > Oxygen XML Editor > XML > XML catalog**.

An example is evaluating 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?recurse=yes;select=*.xml;
  parser=ro.sync.xml.parser.CatalogEnabledXMLReader"))
```

When you run an XPath expression, Oxygen XML Editor plugin displays the results of its execution in the **Results View**. [The Results View](#) on page 47. This view contains five columns:

- Description - holds the result that Oxygen XML Editor plugin 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 - holds the location of the result in the document.

To arrange the results depending on a column click on its header. If no information regarding location is available, Oxygen XML Editor plugin displays **Not available** in the **Location** column. Oxygen XML Editor plugin displays the results in a valid XPath expression format.

```
- /node[value]/node[value]/node[value] -
```



Note: You can stop the current execution of an XPath expression by executing another XPath expression. This is valid when you execute XPath 2.0 and Xpath 3.0 expressions.

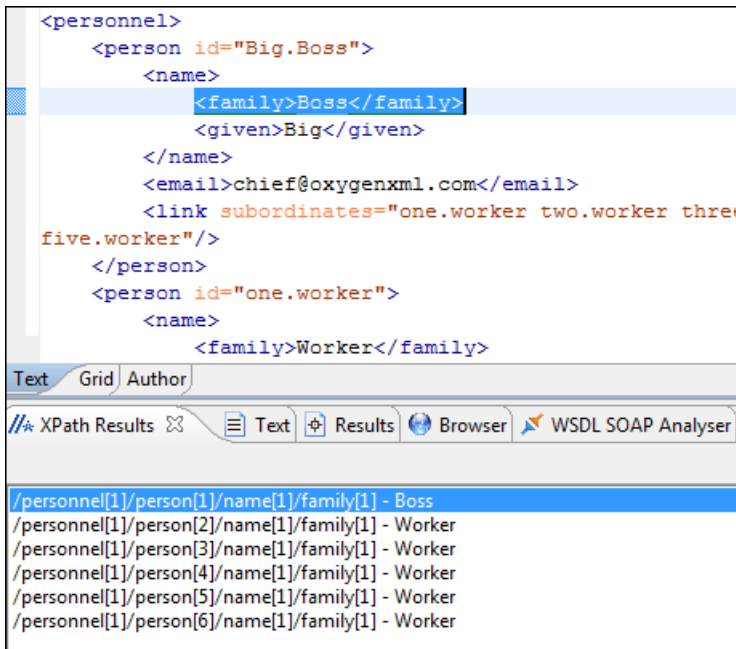


Figure 272: XPath results highlighted in editor panel with character precision

If you use the *grid editor*, when you click a record in the **Results View**, Oxygen XML Editor plugin highlights the entire node.

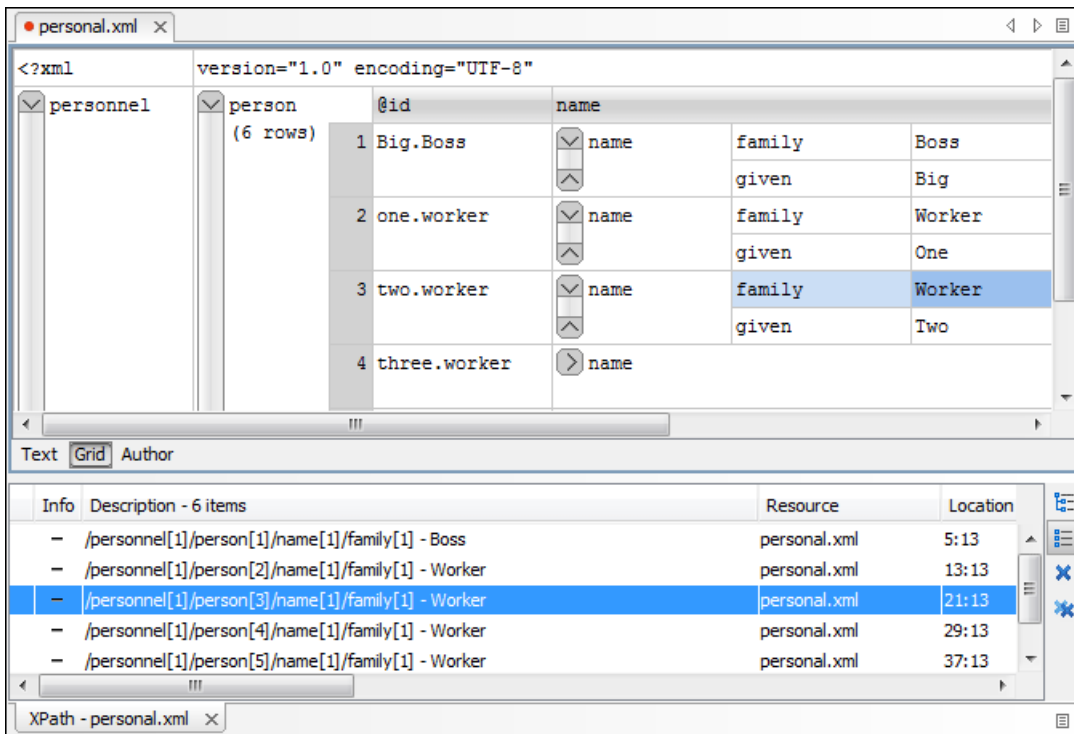


Figure 273: XPath results highlighted in the Grid Editor

The XPath dialog contains a **History** section where you can view the expressions you ran. To remove an expression from the **History** section, right click the expression and select **Remove**. If you select **Remove All**, all the expressions from the History section are removed.

Using XPath with DocBook DTD

The following examples 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 plugin 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.



Important: If you define a default namespace, Oxygen XML Editor plugin 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 dialog 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 dialog.

To define default mappings between prefixes (that you can use in the XPath dialog) and namespace URIs [go to XPath 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. Different results tabs are created for each executed XPath query.

To apply an XPath expression relative to the element on which the caret is positioned, use the following actions:

- **XML editor contextual menu > XML Document > Copy XPath (Ctrl (Meta on Mac OS) + Shift + .)** - copies the XPath expression of the current element or attribute to the clipboard;
- **Paste** - pastes this expression in the dialog;
- add your relative expression in the dialog and execute the resulting complete expression.

The popup menu available in the **Expression** panel of the XPath expressions dialog offers the usual edit actions: **Cut**, **Copy**, **Paste**, **Select All**.


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 plugin 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);
 - a 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 plugin 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 plugin returns a compilation error.

- an  **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;
- a **Favorites** button which 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 plugin automatically groups favorites in folders named after the method of execution;
- a  **Clear content** button that you can use to clear the editing area of the view;
- a  **History** drop-down that keeps a list of the last 15 executed XPath or XQuery expressions;
- a  **Settings** drop-down menu which contains three options:
 -  **Update on caret move** - select this option to allow the **XPath/XQuery Builder** view to display the XPath expression at the current cursor position when you navigate through the document;
 -  **Evaluate as you type** - select this option to allow the **XPath/XQuery Builder** view to evaluate in real time the expression you are composing;
 -  **Options** - opens the Preferences page of the currently selected method.

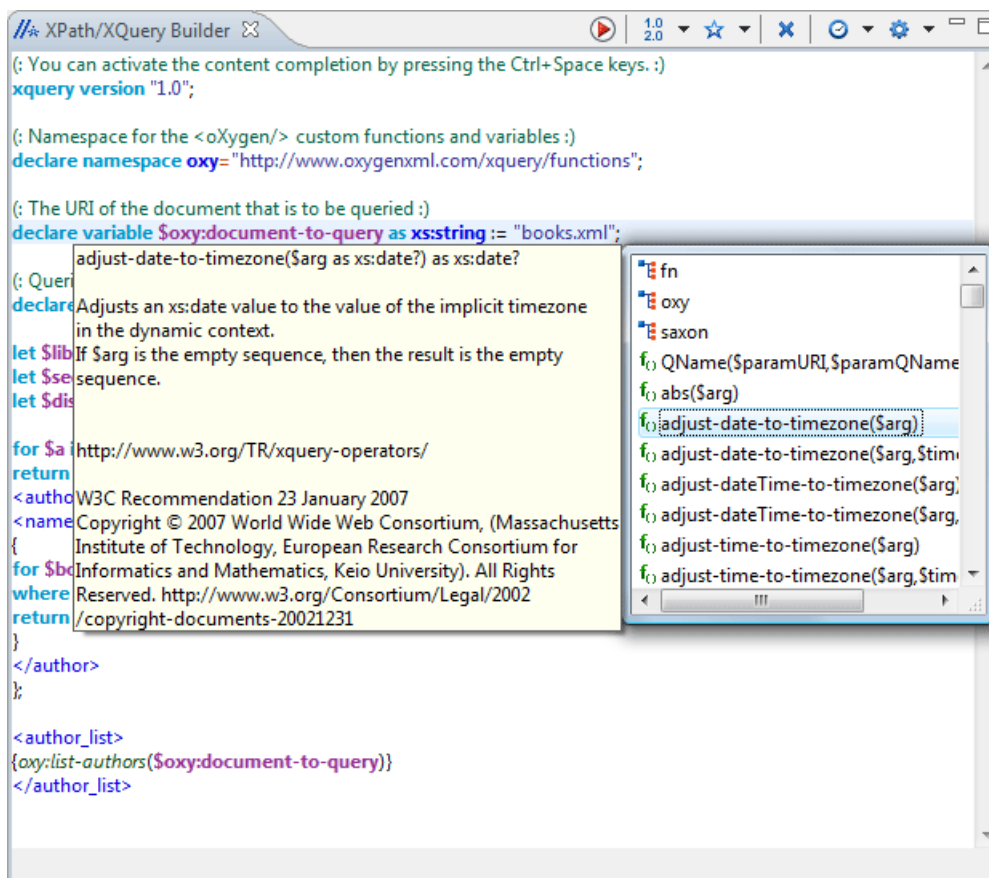



Figure 274: The XPath/XQuery Builder View

When you hover your cursor over the XPath/XQuery version icon , a tooltip is displayed to let you know what engine Oxygen XML Editor plugin currently uses.

While you edit an XPath or XQuery expression, Oxygen XML Editor plugin assists you with the following features:

- **Content Completion Assistant**;
- syntax highlight;
- automatic validation of the expression as you type;



Note: When you type invalid syntax in the XPath/XQuery builder, a red serrated line underlines the invalid fragments.

- method signature balloon, when the cursor is located inside an XQuery function.

The evaluation of the XPath expression tries to resolve the locations of documents referred in the expression through the *XML catalogs*. These catalogs are *configured in the Preferences* pages and *the current XInclude preferences*. For example, these preferences are used when evaluating the XPath 2.0 `collection(URIofCollection)` function.

Working with XQuery

This section explains how to edit and run XQuery queries in Oxygen XML Editor plugin.

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 plugin 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 9.5.0.1 PE or Saxon 9.5.0.1 EE
- the edited file has a validation scenario associated that use as validation engine Saxon 9.5.0.1 PE or Saxon 9.5.0.1 EE
- the validation engine specified in *Preferences* is Saxon 9.5.0.1 PE or Saxon 9.5.0.1 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.

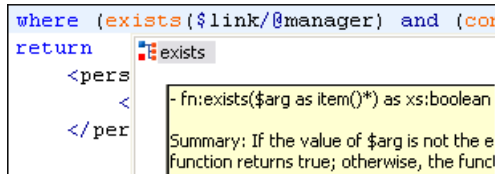


Figure 275: XQuery Content Completion

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 > Other > oXygen > Outline** menu action.

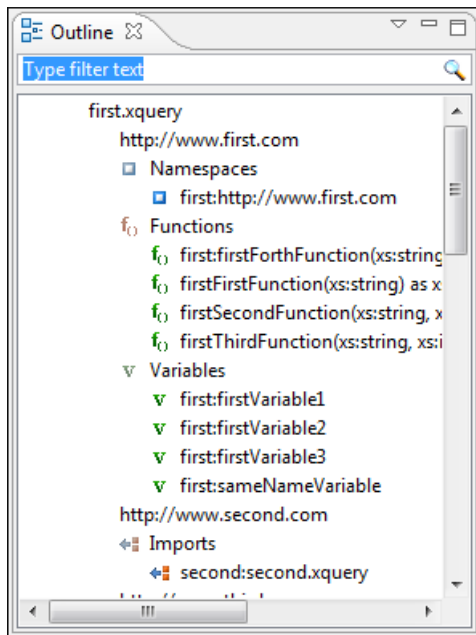



Figure 276: XQuery Outline View

The following actions are available in the **View** menu on the Outline view's action bar:

- **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.

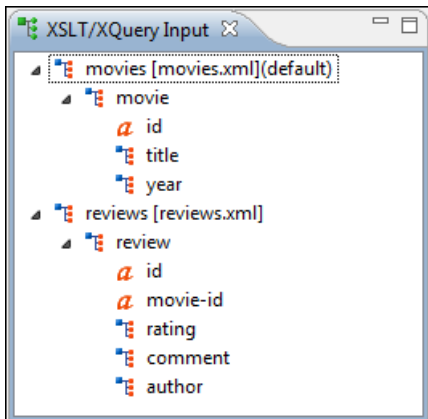


Figure 277: XQuery Input view

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 278: XQuery Input drag and drop popup menu

Select **FLWOR rating** and the result document will be:

```

37 {
38     for $review in doc("reviews.xml")/reviews/review
39     return
40     where ((compare($rev/rating/text(), string($minRating)) eq 0)
41           and ($rev/@movie-id = $movie/@id))
42     return $rev/author
43 }

```

Figure 279: XQuery Input drag and drop result

XQuery Validation

With Oxygen XML Editor plugin, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.5.0.1 PE processor or the 9.5.0.1 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.

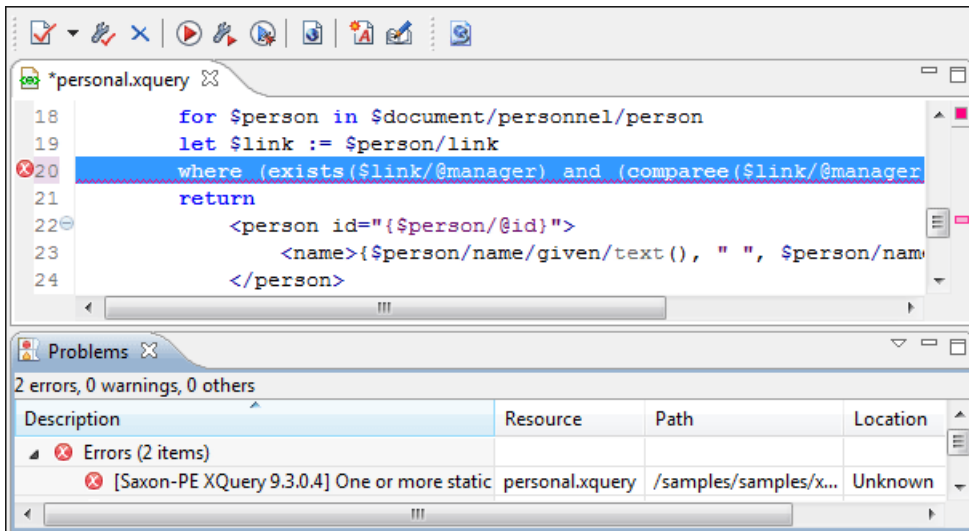


Figure 280: XQuery Validation

Note: In case you choose a processor that does not support XQuery validation, Oxygen XML Editor plugin displays a warning when trying to validate.

When you open an XQuery document from a connection that supports validation (for example MarkLogic, or eXist), by default Oxygen XML Editor plugin 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 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 referred 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 9.5.0.1 HE, Saxon 9.5.0.1 PE, Saxon 9.5.0.1 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 9.5.0.1 EE processor supports also XQuery 3.0 transformations.



Note: In case the XQuery 3.0 support is active, the XQuery Update support is no longer available.

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 Mac OS X. Restart Oxygen XML Editor plugin after configuring the environment variables.
2. Go to menu **Preferences > 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 plugin 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. Go to menu **Preferences > 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.

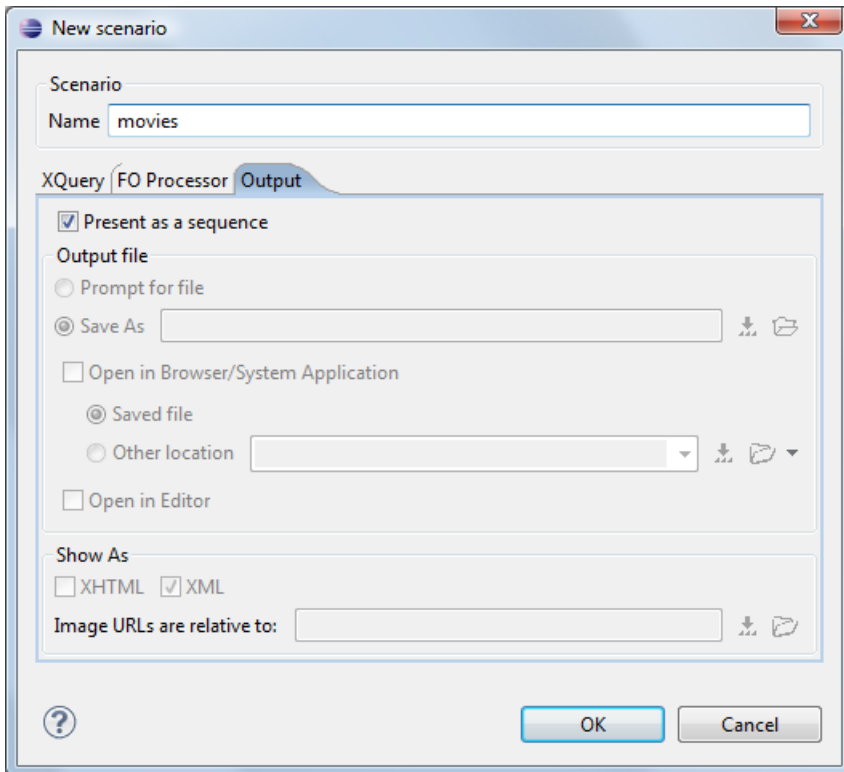


Figure 281: The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.

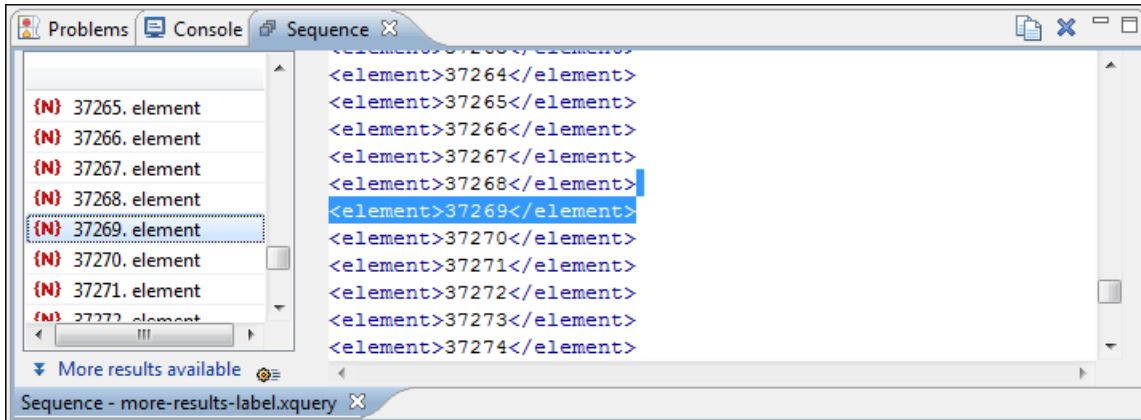


Figure 282: The XQuery transformation result displayed in Sequence view

Advanced Saxon HE/PE/EE Transform Options

The XQuery transformation scenario allows configuring advanced options specific for the Saxon HE (Home Edition) / PE (Professional Edition) / EE (Enterprise Edition) engine. They are the same options as *the ones set in the user preferences* but they are configured as a specific set of transformation options for each transformation scenario. The default values of the options in the transformation scenario are the values *set in the user preferences*. The advanced options specific for Saxon HE / PE / EE are:

- **Use a configuration file** - when enabled, the specified Saxon configuration file are used for determining the Saxon advanced options;
- **Recoverable errors** - policy for handling recoverable errors in the stylesheet. Allows the user to choose how dynamic errors are handled. Either one of the following options can be selected:

- recover silently;
- recover with warnings;
- signal the error and do not attempt recovery.
- **Strip whitespaces** - can have one of the following values:
 - **All** - strips all whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the XML source document;
 - **Ignorable** - strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the XML source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content;
 - **None** - strips no whitespace before further processing.
- **Optimization level** - allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably;
- **Allow calls on extension functions** - when enabled, calling external Java functions is allowed;
- **Validation of the source file** - available only for Saxon EE. It can have three values:
 - **Schema validation** - this mode requires an XML Schema and enables parsing the source documents with schema-validation enabled;
 - **Lax schema validation** - if an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled;
 - **Disable schema validation** - this mode means parsing the source documents with schema validation disabled.
- **Validation errors in the results tree treated as warnings** - available only for Saxon EE. If checked, all validation errors are treated as warnings, otherwise they are treated as fatal;
- **Generate bytecode ("--generateByteCode:(on|off)")** - when 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 treatment. For further details regarding this option, go to <http://www.saxonica.com/documentation/javadoc/>.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - if checked, Saxon EE runs the XQuery transformation with the XQuery 3.0 support;
 - 📌 **Note:** In case the XQuery 3.0 support is active, the XQuery Update support is no longer available.
- **Backup files updated by XQuery ("-backup:(on|off)")** - if checked, backup versions for any XML files updated with XQuery Update are generated.

Updating XML Documents using XQuery

Using the bundled Saxon 9.5.0.1 EE XQuery processor Oxygen XML Editor plugin 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 9.5.0.1 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Editor plugin 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 12

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 9.5.0.1 XSLT engine) and XQuery 1.0 / 3.0 (using Saxon 9.5.0.1 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/included 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.
- **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*

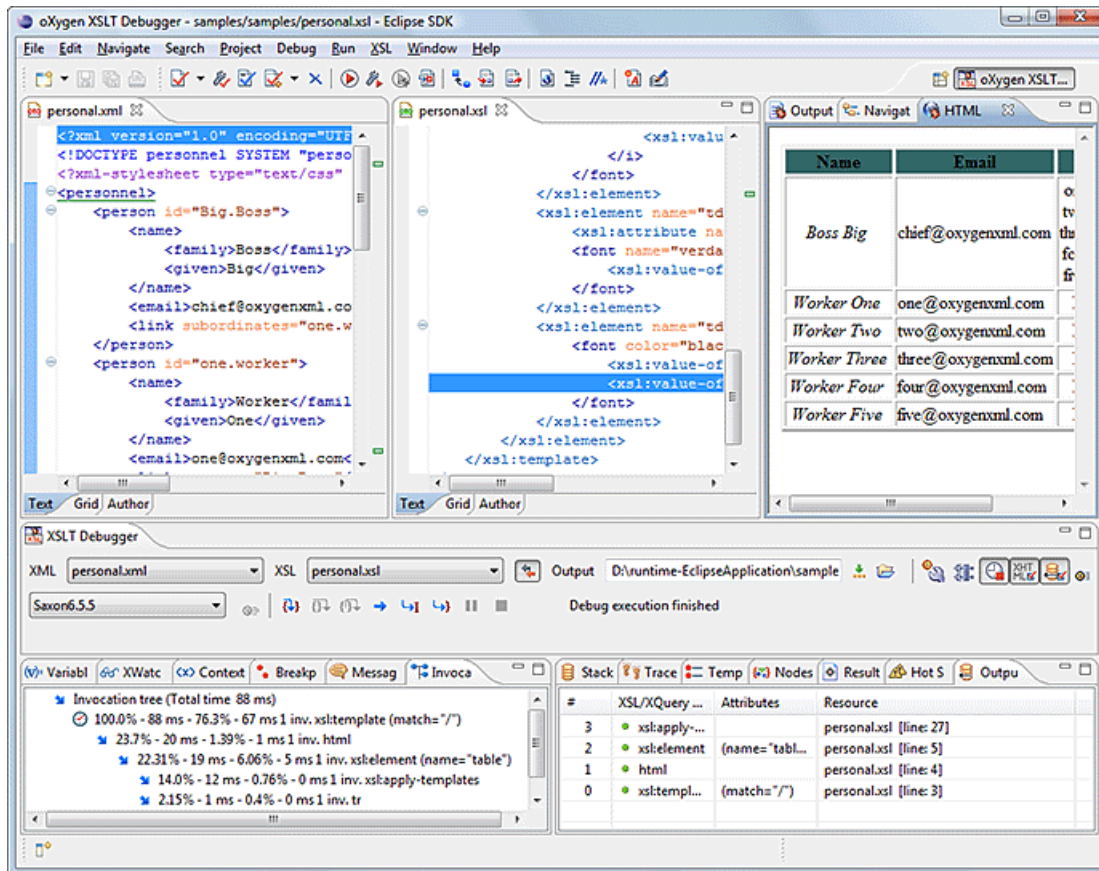


Figure 283: Debugger Mode Interface

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 stylesheets into focus and enables editing without the need to go back to the Editor perspective.

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 284: 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 stylesheets 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 9.5.0.1.
-  **Step into (F7)** - Starts the debugging process and runs until the next stylesheet node or the next XPath 2.0 / 3.0 expression step (next transformation step).
-  **Step over (F8 (Alt+F7 on Mac OS))** - Executes the current stylesheet node (including its sub-elements) and goes to the next node in document order (usually the next sibling of the current node) or to the next step of an XPath 2.0 / 3.0 expression.
-  **Step out (Shift + F7)** - Steps out to the parent node (equivalent to the [Step over](#) on the parent).
-  **Run** - Starts the debugging process. The execution of the process is paused when a breakpoint is encountered. (see [breakpoints](#)).
-  **Run to cursor ((Ctrl (Meta on Mac OS)+ F5)** - 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 ((Alt + F5)** - Runs the transformation until the end, without taking into account enabled breakpoints (if any).
-  **Pause (Shift + F6)** - Interrupts the current transformation. This action is useful for long transformations (DocBook for instance) when you want to find out what point the transformation has reached. The transformation can be resumed after.
-  **Stop (F6)** - Ends the transformation process.
- **Show current execution nodes** - Positions the cursor at the current debug context. 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.

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.

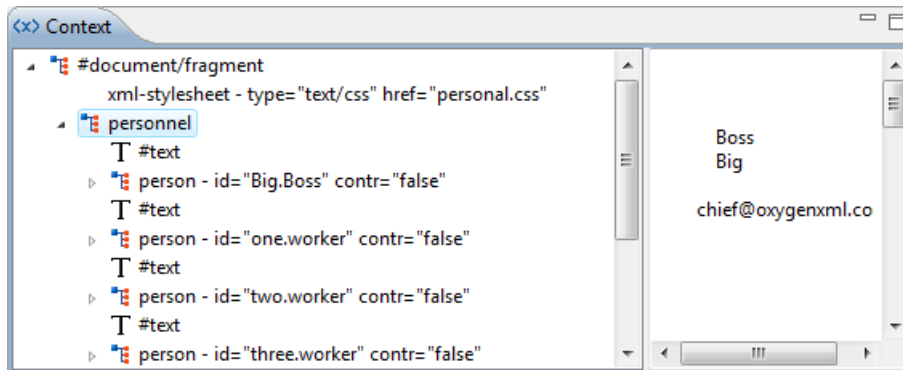


Figure 285: The Context node view

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.

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 plugin supports you with syntax highlight and [content completion assistance](#).

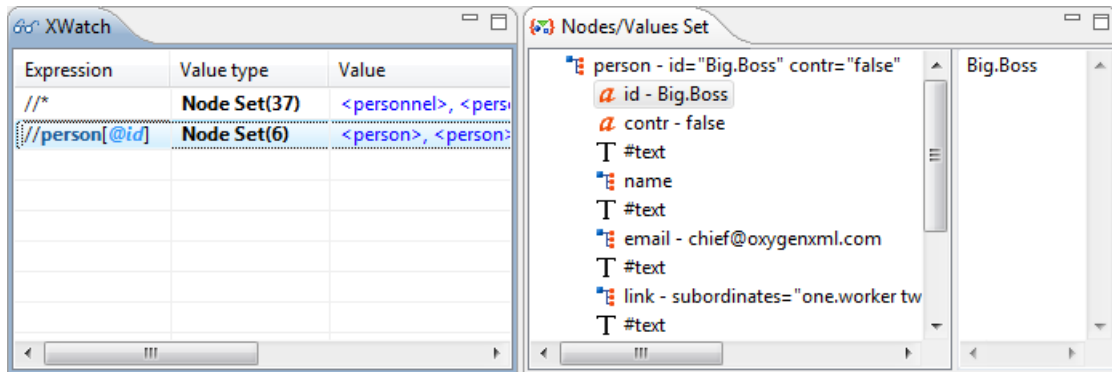


Figure 286: The XPath watch view

Table 8: 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 the possible values in the section Variables View on page 610). 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 referring to variables 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.
- To delete an expression, click on its **Expression** column and delete its content.
- If the expression result type is a *Node Set*, click it (**Value** column) and its value is displayed in the [Nodes/Values Set view](#).
-

Breakpoints View

This view lists all breakpoints set on opened documents. Once you set a breakpoint it is automatically added in this list. Breakpoints can be set in XSLT/XQuery documents and in XML documents for XSLT debugging sessions. A breakpoint can have an associated break conditions which represent XPath expressions 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 stops the execution of the Debugger only if the breakpoint condition is evaluated to **true**.

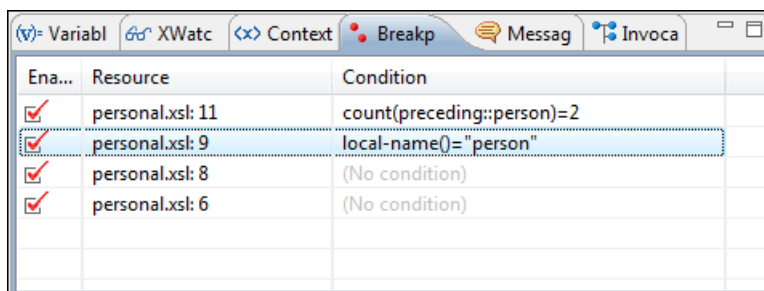


Figure 287: The Breakpoints View

Table 9: Breakpoints columns

Column	Description
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.
Condition	XSLT/XQuery expression to be evaluated during debugging. The expression will be evaluated at every debug step.



Important: Not all set breakpoints are valid. You should check that your breakpoint is valid:

- For example if the breakpoint is set on an empty line or commented line or the line is not reached by the processor (no template to match it, line containing only an end tag), that breakpoint is invalid.
- Clicking a record highlights the breakpoint line into the document.
- The breakpoints list is not deleted at the end of transformation (it is preserved between debugging sessions).

The following actions are available on the table's contextual menu:

- **Go to** - Moves the cursor on the breakpoint's source.
- **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.

Message	Terminate	Resource
Message 1	no	personal.xml [line: 8]
Message 2	no	personal.xml [line: 12]
Message 3	no	personal.xml [line: 29]

Figure 288: The Messages View**Table 10: 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 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 plugin 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.

#	XML/XSL/XQ...	Attributes	Resource
4	xsl:message	(terminate="...	file:D:/runtime-New_configuration/Oxy...
3	xsl:element	(name="tabl...	file:D:/runtime-New_configuration/Oxy...
2	html		file:D:/runtime-New_configuration/Oxy...
1	xsl:template	(match="/")	file:D:/runtime-New_configuration/Oxy...
0	#document		file:D:/runtime-New_configuration/Oxy...

Figure 289: 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 11: 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 #document .
Attributes	Attributes of the node (a list of <code>id="value"</code> pairs).
Resource	Resource file where the node is located.



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.

#	XML/XSL/XQuery Node	Attributes	Resource
8	xsl:value-of	(select="email/text()")	file:/D:/runtime-Eclipse
7	font	(name="verdana") (size="3")	file:/D:/runtime-Eclipse
6	xsl:element	(name="td")	file:/D:/runtime-Eclipse
5	xsl:element	(name="tr")	file:/D:/runtime-Eclipse
4	xsl:template	(match="//person")	file:/D:/runtime-Eclipse
3	xsl:apply-templates		file:/D:/runtime-Eclipse
2	xsl:element	(name="table")	file:/D:/runtime-Eclipse
1	html		file:/D:/runtime-Eclipse
0	xsl:template	(match="/")	file:/D:/runtime-Eclipse

Figure 290: 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 12: 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.
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.

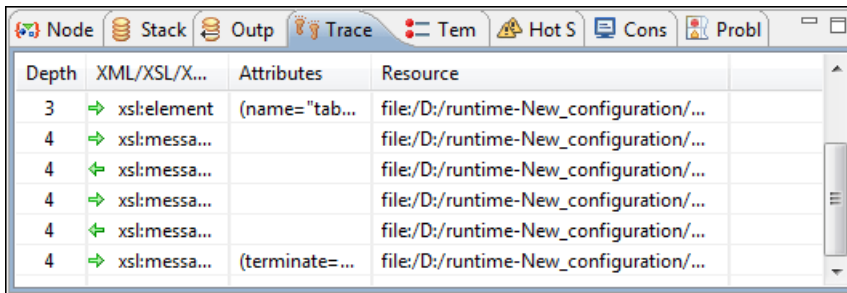


Figure 291: The Trace History View

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 13: 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.

! **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.

Match	Hits	Pr...	M...	N...	Resource
//person	4	N/A	N/A	N/A	file:D:/runtime-New_configuration
/	1	N/A	N/A	N/A	file:D:/runtime-New_configuration

Figure 292: 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 14: Templates columns

Column	Description
Match	The <code>match</code> 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 <code>mode</code> attribute of the <code>xsl:template</code> .
Name	The <code>name</code> attribute of the <code>xsl:template</code> .
Resource	The resource file where the template is located.



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.

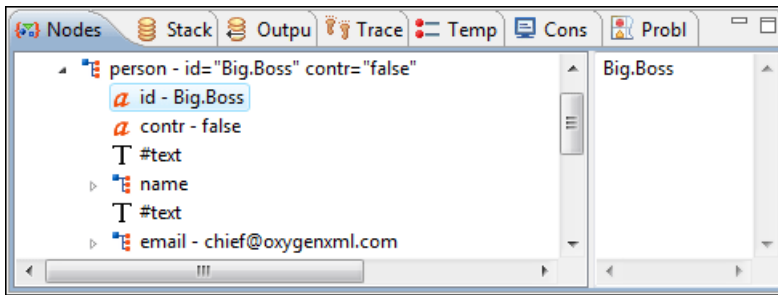


Figure 293: The Node Set view

The nodes / values set 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.

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 plugin uses the following icons to differentiate variables and parameters:

- **V** - Global variable.
- **{V}** - Local variable.
- **P** - Global parameter.
- **{P}** - 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.

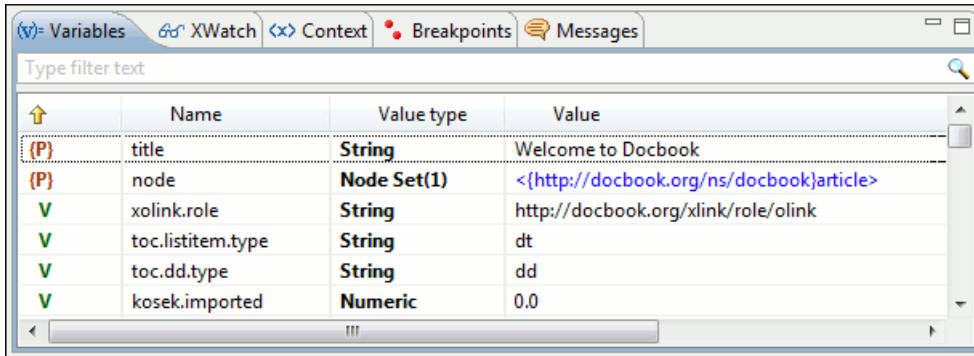


Figure 294: The Variables View

Table 15: 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.

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 Oxygen XML Editor plugin XML perspective switch to the Oxygen XML Editor plugin XSLT Debugger perspective or the Oxygen XML Editor plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):
 - Menu **Window > Open Perspective > Other ... > Oxygen XSLT Debugger**
 - 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**
8. 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 plugin XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure breakpoints persistence between work sessions, they are saved at project level. You can set maximum 100 breakpoints per project.

Inserting Breakpoints

To insert a breakpoint, follow these steps:

1. Click the line where you want to insert the breakpoint in the XML source document or the XSLT / XQuery document.
You can set breakpoints on XML source only for XSLT debugging sessions.
Breakpoints are created on the ending line of a start tag automatically, even if you click a different line.
2. Click the left side stripe of the editor panel.

Removing Breakpoints

Only one action must be executed for removing a breakpoint:



Click the breakpoint icon on the left side stripe of the editor panel. As alternative go to menu **Edit > Breakpoints > Remove All**.

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.

1. Switch to the Oxygen XML Editor plugin XSLT Debugger perspective or Oxygen XML Editor plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):
 - Go to menu **Window > Open Perspective > Other ... > Oxygen XSLT Debugger**
 - Go to menu . 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 **Output** view of the Oxygen XML Editor plugin XSLT Debugger or Oxygen XML Editor plugin XQuery Debugger perspectives.

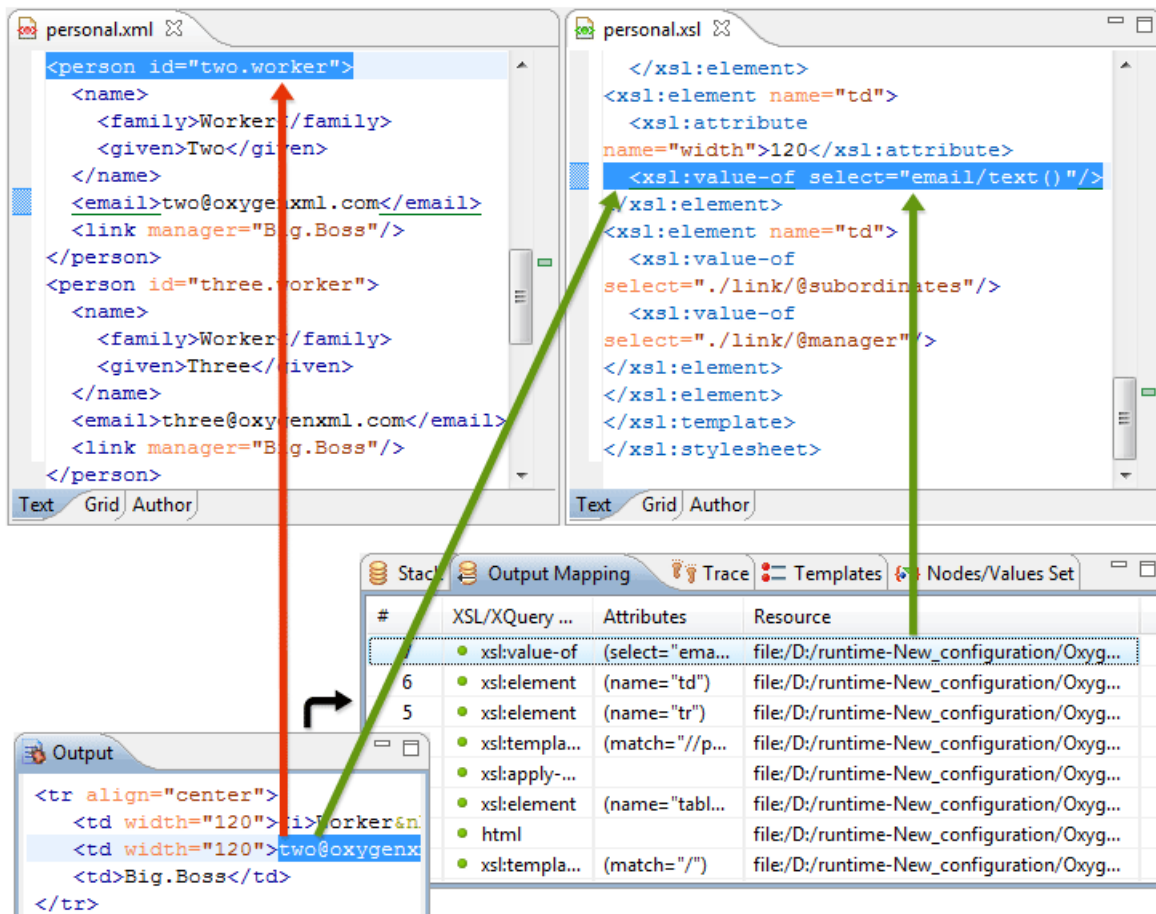


Figure 295: Text Output to Source Mapping

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-install-folder]/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 Oxygen 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 9.5.0.1 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 9.5.0.1 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 9.5.0.1 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 13

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 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.

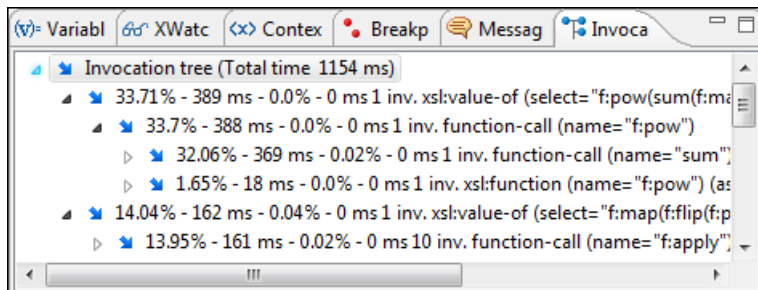




Figure 296: Invocation tree view

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](#).

Instruction	Percentage	Time	Calls
85 Hotspots			
xsl:choose	8.38%	151 ms	698
8.38% - 151 ms - 698 inv. let (name="vdiffResult")			
8.38% - 151 ms - 698 inv. let (name="vnewResult")			
8.38% - 151 ms - 698 inv. let (name="vnewElem")			
8.38% - 151 ms - 698 inv. let (name="vnew")			
8.38% - 151 ms - 698 inv. xsl:function (name="int:InIter")			
let (name="vdiffResult")			698
xsl:apply-templates (select="\$pFunc") (mode="f:FXSL")			695
function-call (name="int:InIter")	5.75%	103 ms	632

Figure 297: 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.

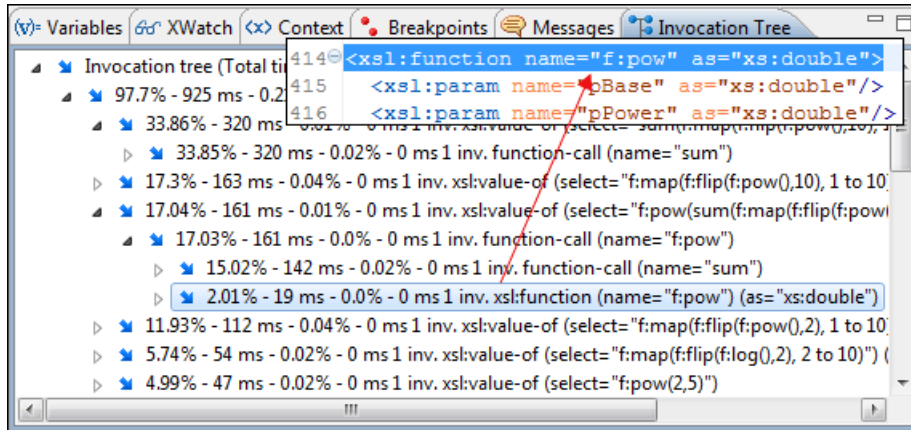


Figure 298: Source backmapping

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 plugin 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 plugin 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 plugin distribution (see the subfolder `frameworks/profiler/` of the Oxygen XML Editor plugin installation folder) 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://oxygenxml.com/demo/XSLT_Profiling.html.

Chapter 14

Working with Archives


Topics:

- [Browsing and Modifying Archive Structure](#)
- [Working with EPUB](#)
- [Editing Files From Archives](#)

Oxygen XML Editor plugin 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 navigate archives in the **Archives Browser** either by opening them from the **Navigator** or by using the integration with the Eclipse File System. For the EFS (Eclipse File System) integration you must right click the archive in the **Navigator** and choose **Expand Zip Archive**. All the standard Eclipse **Navigator** actions are available on the mounted archive. If you decide to close the archive you can use the **Collapse ZIP Archive** action located in the contextual menu for the expanded archive. Any file opened from the archive expanded in the EFS will be closed when the archive is unmounted.

If you open an archive as an Eclipse editor, the archive will be unmounted when the editor is closed.

! **Important:** If a file is not recognized by Oxygen XML Editor plugin as a supported archive type, you can add it from the [Archive preferences page](#).

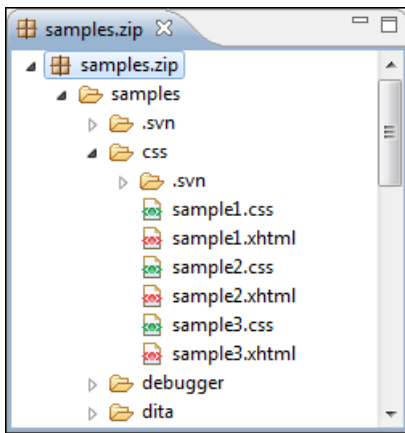


Figure 299: Browsing an Archive

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.
- **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:

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.
- **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 Mac OS X, there is also available the **Add file...** action, which allows you to add one file at a time.

-  **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
 - **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 plugin supports both EPUB 2.0 and EPUB 3.0.

Opening an EPUB file exposes all its internal components:

- document content (XHTML and image files);
- packaging files;
- container files.

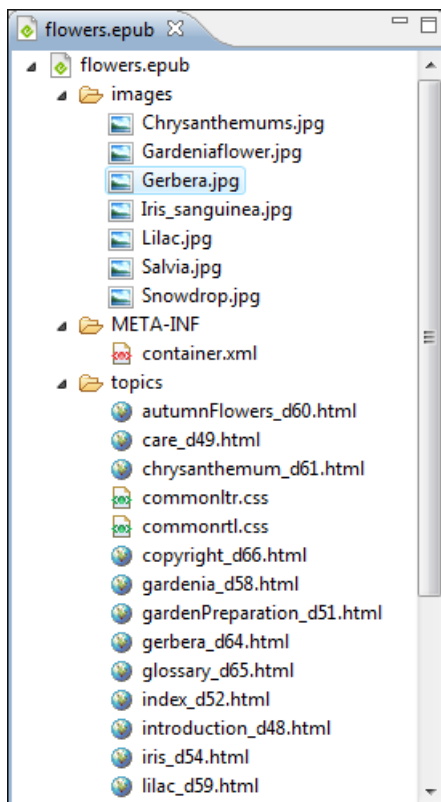






Figure 300: EPUB file displayed in Eclipse

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 plugin 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 plugin, go to <http://oxygenxml.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 (Meta on Mac OS) + N** 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 plugin 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 dialog pop-up from the [Archive 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 15

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 it is supported by all major database systems. Oxygen XML Editor plugin offers the means of managing the interaction with some of the widely used databases, both relational ones and Native XML Databases. By interaction, one should understand browsing, querying, SQL execution support, content editing, importing from databases, 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 plugin offers support for the following relational databases: IBM DB2, JDBC-ODBC Bridge, MySQL, Microsoft SQL Server, Oracle 11g:

- browsing the tables of these types of database in the **Data Source Explorer** view
- executing SQL queries against them
- calling stored procedures with input and output parameters

To watch our video demonstration about the integration between the relational databases and Oxygen XML Editor plugin, go to 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.

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 the following:

1. Go to **Preferences > Data Sources**.
2. Click the **+** **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

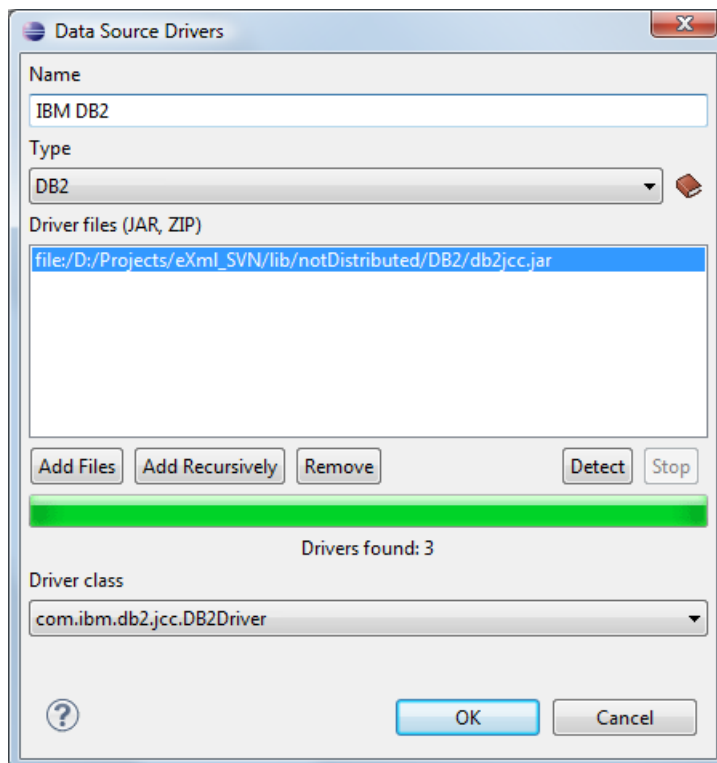


Figure 301: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **DB2** in the driver type combo box.
5. Add the driver files for IBM DB2 using the **Add** button.

The IBM DB2 driver files are:

- db2jcc.jar;
- db2jcc_license_cisuz.jar;
- db2jcc_license_cu.jar.

In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing IBM DB2 databases in Oxygen XML Editor plugin.

6. Select the most suited **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 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 the following:

1. Go to **Preferences > Data Sources**.
2. Click the **+** **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

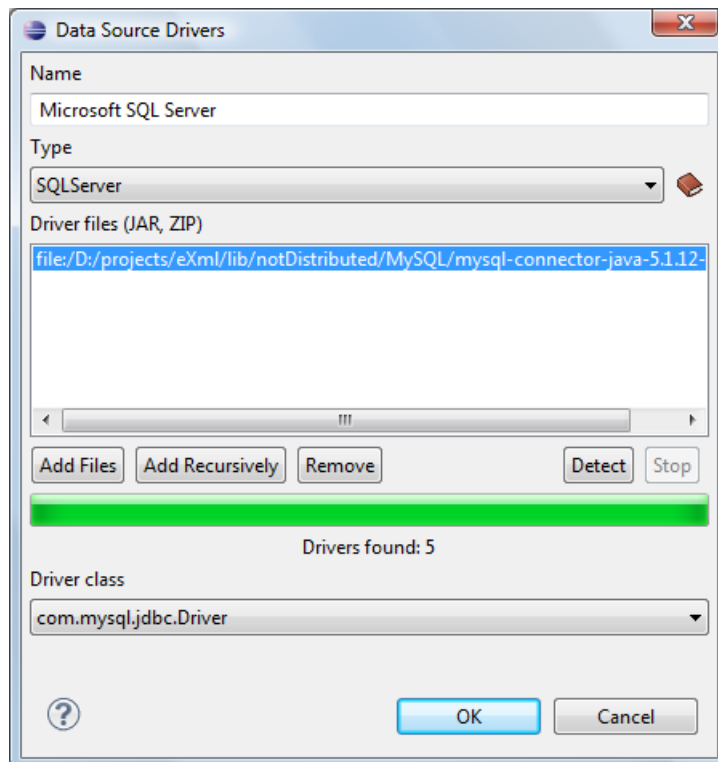


Figure 302: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **SQLServer** in the driver type combo box.

5. Add the Microsoft SQL Server driver file using the **Add** button.

The SQL Server driver file is called `sqljdbc.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing Microsoft SQL Server databases in Oxygen XML Editor plugin.

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a Generic JDBC Data Source

By default, Oxygen XML Editor plugin contains a generic JDBC data source called **JDBC-ODBC Bridge**. Oxygen XML Editor plugin can display and edit XML data stored in PostgreSQL and Microsoft SQL Server databases accessible through a JDBC 4 driver. To do this, configure a **Generic JDBC** data source that uses a JDBC 4 driver. The following procedure shows you how to configure a generic JDBC data source:

1. Go to **Preferences > Data Sources**.
2. Click the **+** **New** button in the **Data Sources** panel.

The following dialog is displayed:

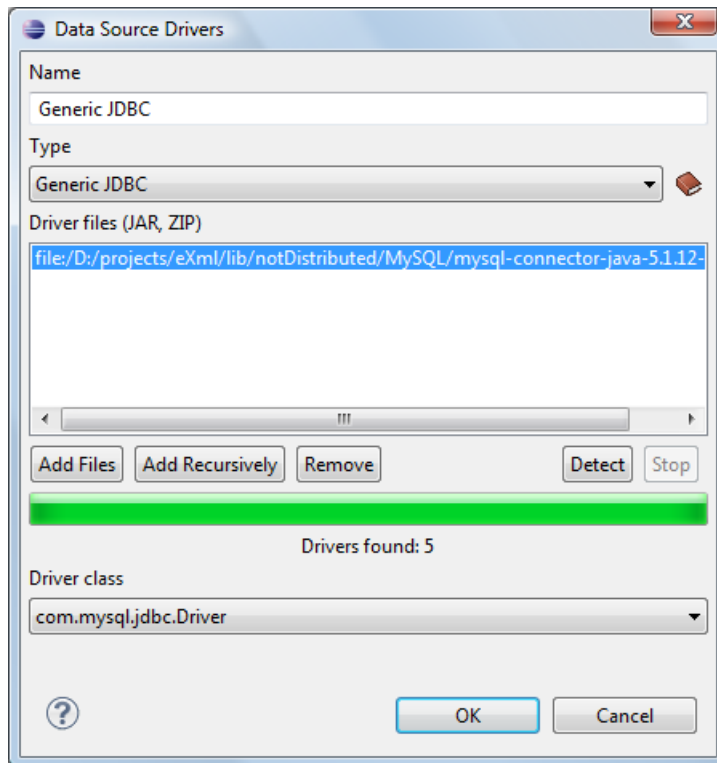


Figure 303: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Generic JDBC** in the driver type combo box.
5. Add the driver file(s) using the **Add** button.
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a MySQL Data Source

Older versions of Oxygen XML Editor plugin (up to version 11.2) include a built-in type of data sources called **MySQL** based on the JDBC driver for the MySQL 4 server. That type of data source is still available but is marked *outdated*

because it does not support more recent versions of the MySQL server (starting from version 5.0) and it will be removed in a future version of Oxygen XML Editor plugin. To connect to a MySQL server, create a data source of type Generic JDBC based on *the MySQL JDBC driver available on the MySQL website*. The following steps describe how you can configure such a data source:

1. Go to **Preferences > Data Sources**.
2. Click the **+** **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

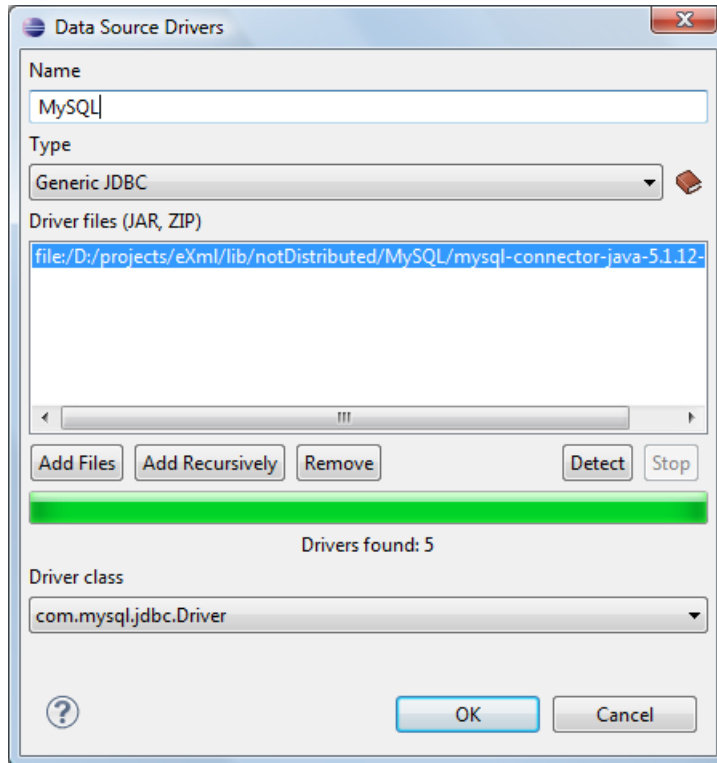


Figure 304: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Generic JDBC** in the driver type combo box.
5. Add the MySQL 5 driver files using the **Add** button.

The driver file for the MySQL server is called `mysql-com.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing MySQL databases in Oxygen XML Editor plugin.

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

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 the following:

1. Go to **Preferences > Data Sources**.
2. Click the **+** **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

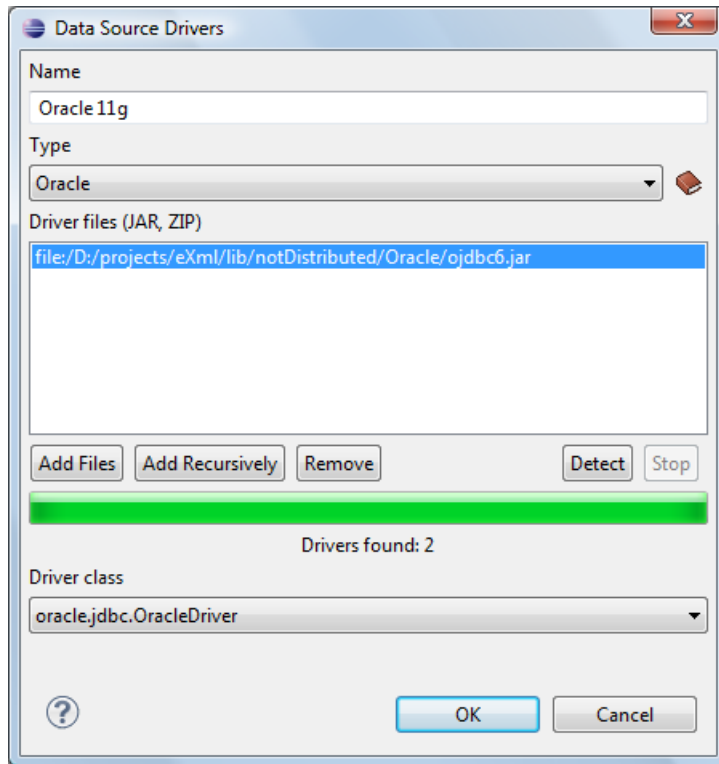


Figure 305: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Oracle** in the driver type combo box.
5. Add the Oracle driver file using the **Add** button.

The Oracle driver file is called `ojdbc5.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing Oracle databases in Oxygen XML Editor plugin.

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a PostgreSQL 8.3 Data Source

The steps for configuring a data source for connecting to a PostgreSQL server are the following:

1. Go to **Preferences > Data Sources**.
2. Click the **+ New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

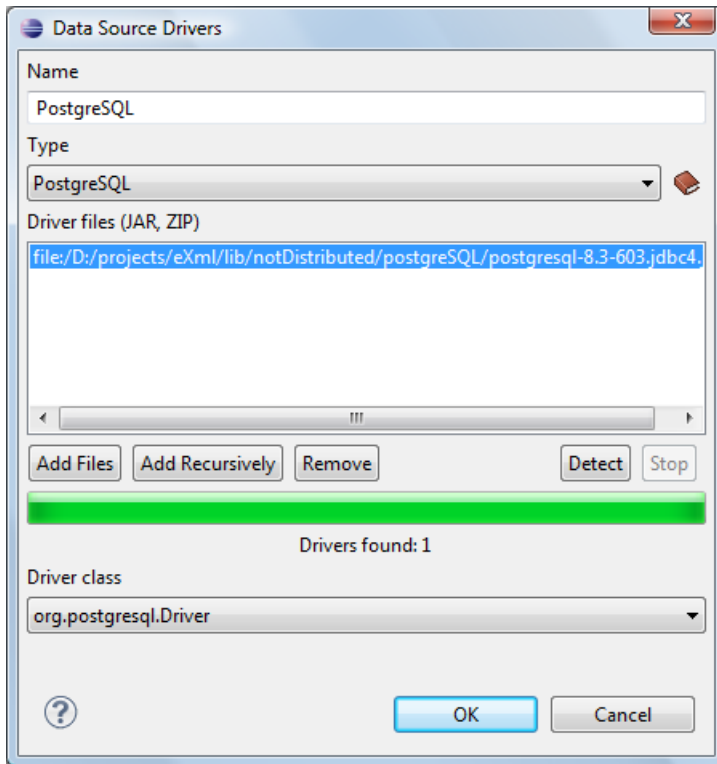


Figure 306: Data Source Drivers Configuration Dialog

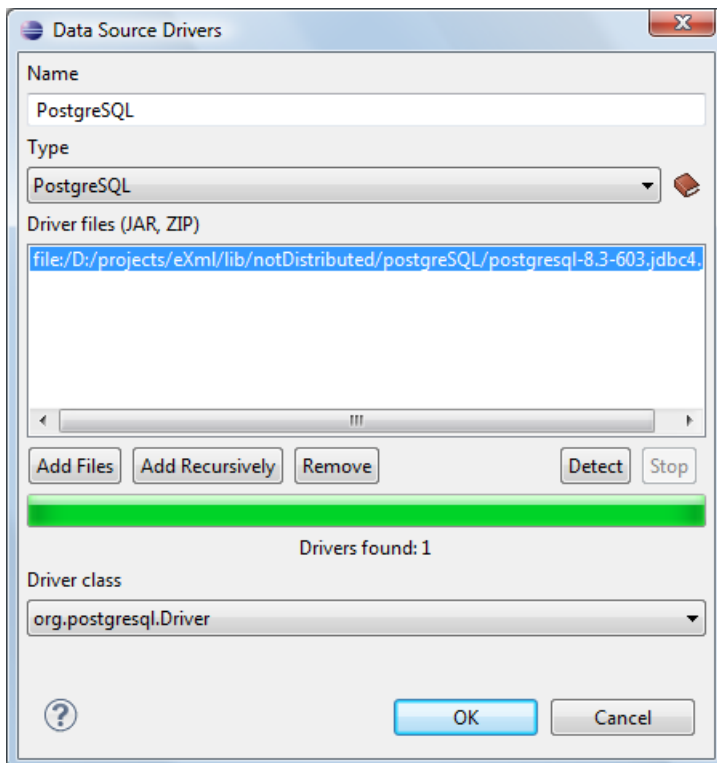


Figure 307: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **PostgreSQL** in the driver type combo box.

5. Add the PostgreSQL driver file using the **Add** button.

The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing PostgreSQL databases in Oxygen XML Editor plugin.

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

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. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog for configuring a database connection will be displayed.

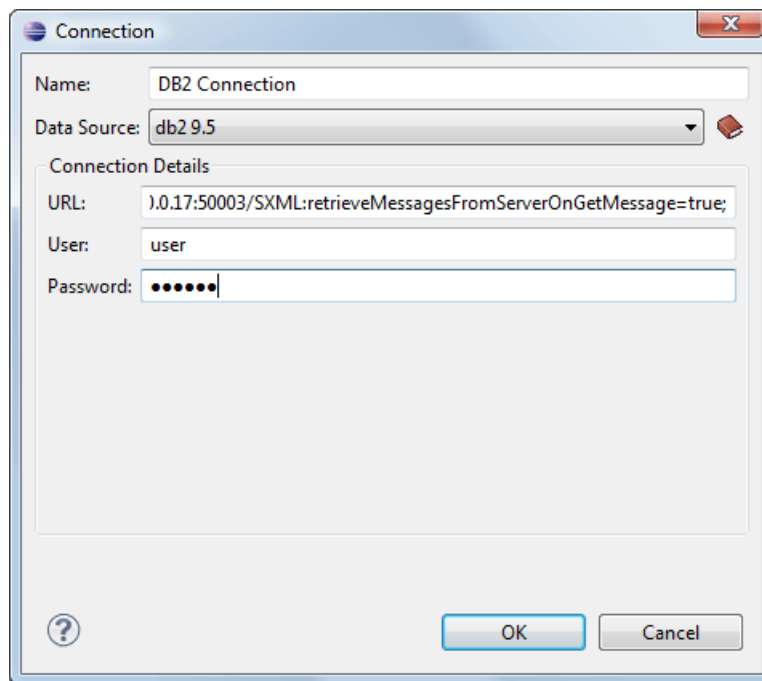


Figure 308: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select an IBM DB2 data sources in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL to the installed IBM DB2 engine.
 - b) Fill-in the user name to access the IBM DB2 engine.
 - c) Fill-in 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 a JDBC-ODBC Connection

To configure a connection to an ODBC data source, follow these steps:

1. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog for configuring a database connection will be displayed.



Figure 309: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select JDBC-ODBC bridge in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the ODBC source.
 - b) Fill-in the user name of the ODBC source.
 - c) Fill-in the password of the ODBC source.
6. Click the **OK** button to finish the configuration of the database connection.

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. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog for configuring a database connection will be displayed.

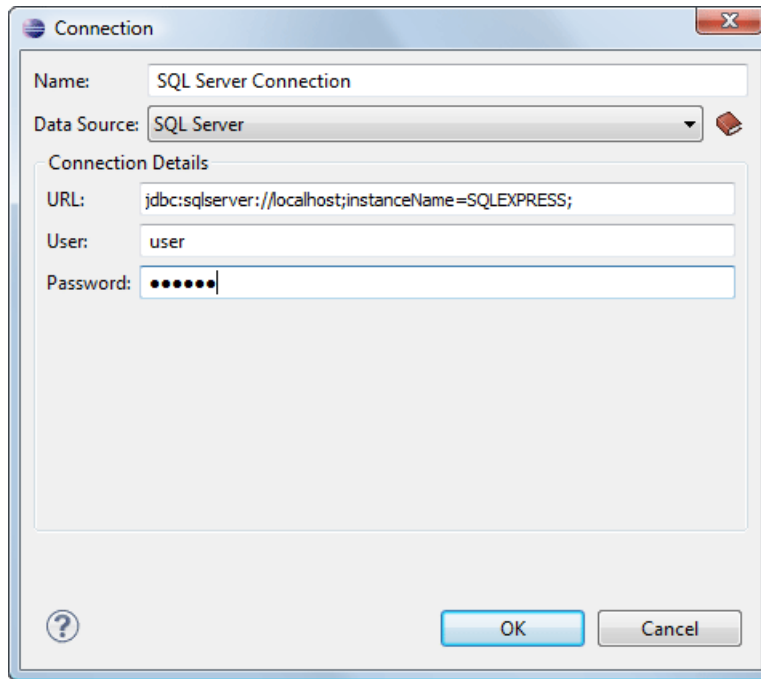


Figure 310: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select a SQL Server data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in 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, so the URL will look like:

```
jdbc:sqlserver://localhost;instanceName=SQLEXPRESS;integratedSecurity=true;
```
 - b) Fill-in the user name for the connection to the SQL Server.
 - c) Fill-in the password for the connection to the SQL Server.
6. Click the **OK** button to finish the configuration of the database connection.



Note: For integrated authentication, leave the **User** and **Password** fields empty.

How to Configure a MySQL Connection

To configure a connection to a MySQL server, follow these steps:

1. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog for configuring a database connection will be displayed.

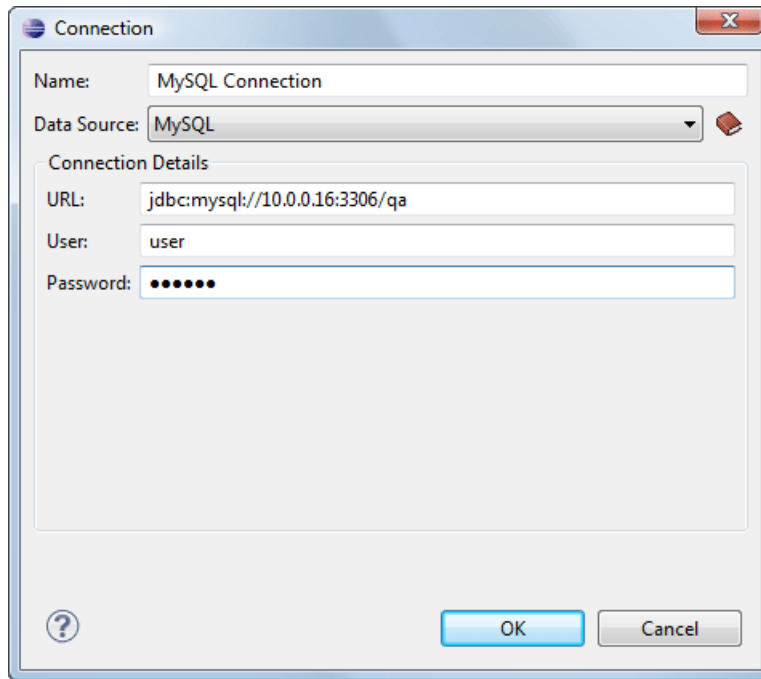


Figure 311: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select a MySQL data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the MySQL server.
 - b) Fill-in the user name for the connection to the MySQL server.
 - c) Fill-in the password for the connection to the MySQL server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a Generic JDBC Connection

To configure a connection to a generic JDBC database, follow these steps:

1. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.
The dialog for configuring a database connection will be displayed.
3. Enter a unique name for the connection.
4. Select a generic JDBC data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the generic JDBC database, with the following format: `jdbc: <subprotocol>: <subname>`.
 - b) Fill-in the user name for the connection to the generic JDBC database.
 - c) Fill-in 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 an Oracle 11g Connection

Available in the Enterprise edition only.

The steps for configuring a connection to an Oracle 11g server are the following:

1. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog for configuring a database connection will be displayed.

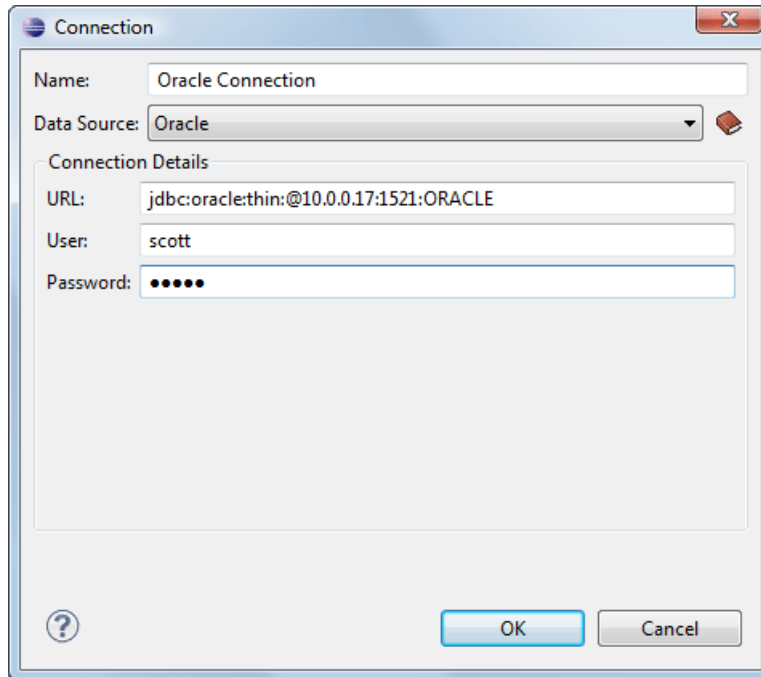


Figure 312: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select an Oracle 11g data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the Oracle server.
 - b) Fill-in the user name for the connection to the Oracle server.
 - c) Fill-in 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 a PostgreSQL 8.3 Connection

The steps for configuring a connection to a PostgreSQL 8.3 server are the following:

1. Go to **Preferences > Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog for configuring a database connection will be displayed.

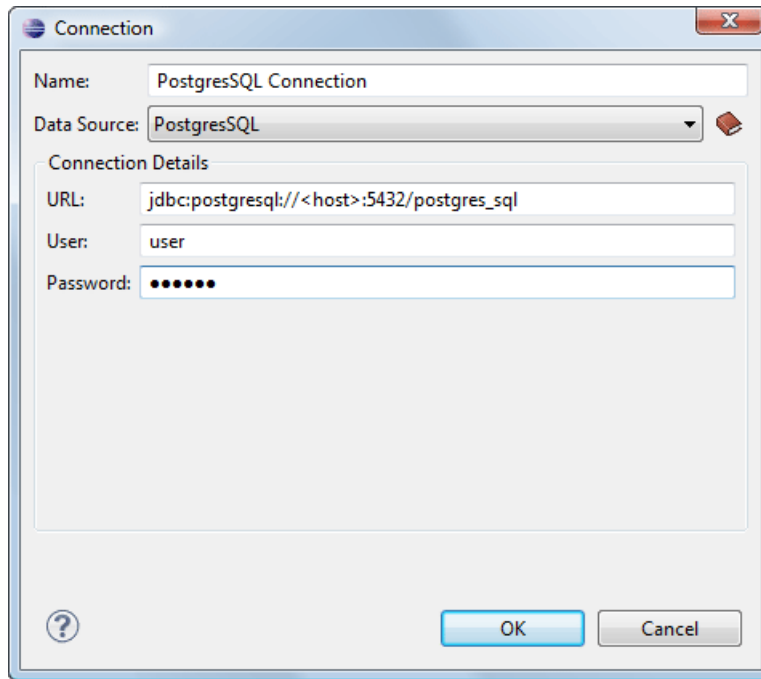


Figure 313: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select a PostgreSQL 8.3 data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the PostgreSQL 8.3 server.
 - b) Fill-in the user name for the connection to the PostgreSQL 8.3 server.
 - c) Fill-in the password for the connection to the PostgreSQL 8.3 server.
6. Click the **OK** button to finish the configuration of the database connection.

Resource Management

This section explains the resource management actions for relational databases.

Data Source Explorer View

This view presents in a tree-like fashion the database connections configured in the **Options > Preferences > Data Sources** preferences page. You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. Oxygen XML Editor plugin supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.

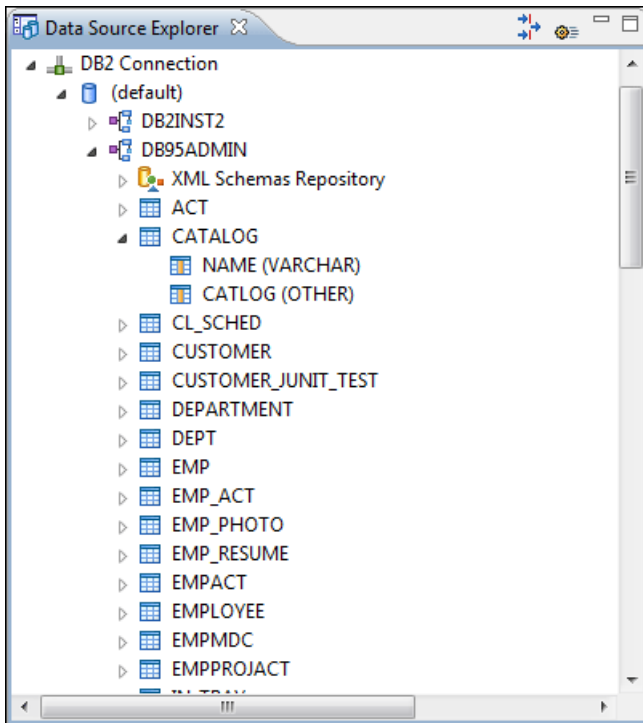


Figure 314: Data Source Explorer View

The following objects are displayed in the **Data Source Explorer** view:

- **Connection** ;
- **Catalog (Collection)**;
- **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 further sub-collections. There are two types of resources:

- **XML resource** - an XML document or a document fragment, selected by a previously executed XPath query.
- **non XML resource**


Note: For some connections you can add or move resources into a container by dragging them from:



- **Project view**;
- the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example);
- or from another database container.

The following actions are available in the view's toolbar:


- The **Filters** button opens the **Data Sources / Table Filters Preferences page**, allowing you to decide which table types will be displayed in the **Data Source Explorer** view.
- The **Configure Database Sources** button 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 of the tree from the **Data Source Explorer** view contains the following actions:


-  **Refresh** - performs a refresh of the selected node's subtree.
- **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 of the tree from the **Data Source Explorer** view contains the following actions:


-  **Refresh** - Performs a refresh of the selected node's subtree.




Actions Available at Schema Level in Data Source Explorer View

The contextual menu of a  **Schema** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.

Actions Available at Table Level in Data Source Explorer View


The contextual menu of a  **Table** node of the tree from the **Data Source Explorer** view contains the following actions:


-  **Refresh** - Performs a refresh of the selected node's subtree.
-  **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 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 of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **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 will be 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 users' database schemas.

IBM DB2's XML Schema Repository Level



The contextual menu of a  **XML Schema Repository** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Register** - Opens a dialog for adding a new schema file in the XML Schema repository. In this dialog 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 into relational tables. Which parts map to which tables and columns is specified into the schema annotations.


Schema dependencies management is done by using the **Add** and **Remove** buttons.

The actions available at  **Schema** level are the following:

-  **Refresh** - Performs a refresh of the selected node (and it's subtree).
- **Unregister** - Removes the selected schema from the XML Schema Repository.
-  **View** - Opens the selected schema in Oxygen XML Editor plugin.

Microsoft SQL Server's XML Schema Repository Level

The contextual menu of a  **XML Schema Repository** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **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 the following:



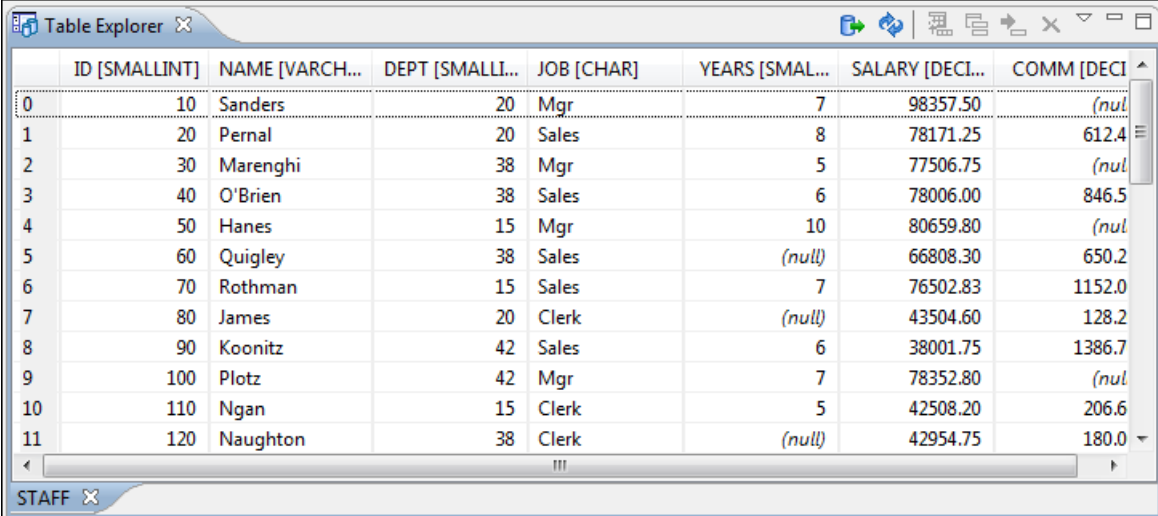
-  **Refresh** - Performs a refresh of the selected node (and it's subtree).
- **Add** - Adds a new schema to the XML Schema files.
- **Unregister** - Removes the selected schema from the XML Schema Repository.
-  **View** - Opens the selected schema in Oxygen XML Editor plugin.

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 a cell's content, double click

it and start typing. When editing is finished, Oxygen XML Editor plugin will try to update the database with the new cell content.




	ID [SMALLINT]	NAME [VARCHAR...]	DEPT [SMALL...]	JOB [CHAR]	YEARS [SMAL...]	SALARY [DECI...]	COMM [DECI...]
0	10	Sanders	20	Mgr	7	98357.50	(nul
1	20	Pernal	20	Sales	8	78171.25	612.4
2	30	Marenghi	38	Mgr	5	77506.75	(nul
3	40	O'Brien	38	Sales	6	78006.00	846.5
4	50	Hanes	15	Mgr	10	80659.80	(nul
5	60	Quigley	38	Sales	(null)	66808.30	650.2
6	70	Rothman	15	Sales	7	76502.83	1152.0
7	80	James	20	Clerk	(null)	43504.60	128.2
8	90	Koonitz	42	Sales	6	38001.75	1386.7
9	100	Plotz	42	Mgr	7	78352.80	(nul
10	110	Ngan	15	Clerk	5	42508.20	206.6
11	120	Naughton	38	Clerk	(null)	42954.75	180.0

Figure 315: 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 (does not belong to 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 will be displayed in the **Table Explorer** view (the **Limit the number of cells** field from the *Data Sources* Preferences page). If a table having more cells than the value set in Oxygen XML Editor plugin's options is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

You will be notified if the value you have entered in a cell is not valid (and thus it cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, an Information dialog will appear, notifying you that the value you have inserted cannot be converted to the SQL type of that field. For example, in the above figure `propID` contains `LONG` values. If a character or string was inserted, you would get the error message that a String value cannot be converted to the requested SQL type (NUMBER).
- If the constraints of the database are not met (like primary key constraints for example), an Information dialog will appear, notifying you of the reason the database has not been updated. For example, if you'd try to set the primary key `propID` for the second record in the table to 10 also, you would get the following message:

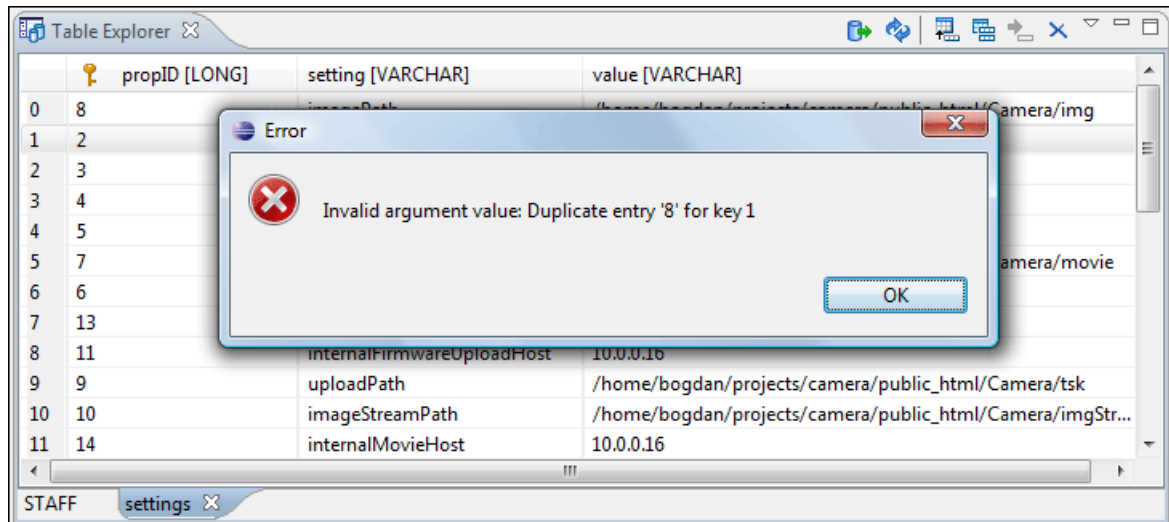


Figure 316: Duplicate entry for primary key

The usual edit actions (**Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo**) are available in the popup menu of the edited cell.

The contextual menu available on every cell has the following actions:

- Set NULL - Sets the content of the cell to (null). This action is disabled for columns that cannot be 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. You should note that the new row will not be inserted in the database table until all conflicts are solved.
- **Commit row** - Commits the selected row.
- **Delete row** - Deletes the selected row.
- **Copy** - Copies the content of the cell.
- **Paste** - Performs paste in the selected cell.

Some of the above actions are also available on the **Table Explorer** toolbar:

- **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 of the selected node's subtree.
- **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. You should note that the new row will not be inserted in the database table until all conflicts are solved.
- **Commit row** - Commits the selected row.
- **Delete row** - Deletes the selected row.

SQL Execution Support

Oxygen XML Editor plugin's support for writing SQL statements includes syntax highlight, folding and drag&drop (DND) 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 (DND) from the **Data Source Explorer** view to the SQL editor allows creating SQL statements quickly by inserting the names of tables and columns in the SQL statements.

1. Configure a database connection (see the procedure specific 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.

DND is available both on the table and on its fields. A popup menu is displayed in the SQL editor.

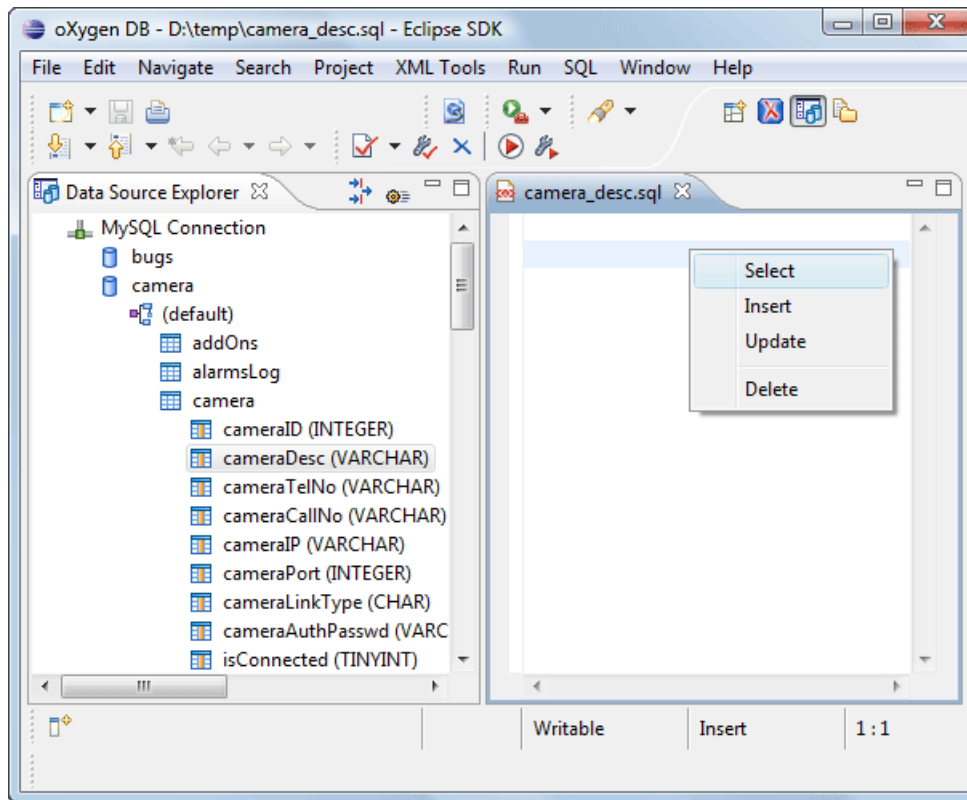


Figure 317: SQL statement editing with DND

4. Select the type of statement from the popup menu.

If you dragged a table depending on your choice, one of the following statements are inserted into the document:

- `SELECT `field1`,`field2`, FROM `catalog`.`table` (for this example: SELECT `DEPT`,`DEPTNAME`,`LOCATION` FROM `camera`.`cameraDesc`)`
- `UPDATE `catalog`.`table` SET `field1`=,`field2`=,.... (for this example: UPDATE `camera`.`cameraDesc` SET `DEPT`=, `DEPTNAME`=, `LOCATION`=)`
- `INSERT INTO `catalog`.`table` (`field1`,`field2`,) VALUES (, ,) (for this example: INSERT INTO `camera`.`cameraDesc` (`DEPT`,`DEPTNAME`,`LOCATION`) VALUES (, ,))`
- `DELETE FROM `catalog`.`table` (for this example: DELETE FROM `camera`.`cameraDesc`)`

If you dragged a column depending on your choice, one of the following statements are inserted into the document:

- `SELECT `field` FROM `catalog`.`table` (for this example: SELECT `DEPT` FROM `camera`.`cameraDesc`)`
- `UPDATE `catalog`.`table` SET `field`= (for this example: UPDATE `camera`.`cameraDesc` SET `DEPT`=)`
- `INSERT INTO `catalog`.`table` (`field1`) VALUES () (for this example: INSERT INTO `camera`.`cameraDesc` (`DEPT`) VALUES ())`
- `DELETE FROM `catalog`.`table` (for this example: DELETE FROM `camera`.`cameraDesc` WHERE `DEPT`=)`


SQL Validation

Currently, SQL validation support is offered for IBM DB2. Please note that if you choose a connection that doesn't support SQL validation you will receive a warning when trying to validate. The SQL document will be validated using the connection from the associated transformation scenario.

Executing SQL Statements

The steps for executing an SQL statement on a relational database are the following:


1. Configure a *transformation scenario* from the  **Configure Transformation Scenario** button from the **Transformation** toolbar.

A SQL transformation scenario needs a database connection. You can configure a connection from the  **Preferences** button from the scenario dialog.

The dialog that appears contains the list of existing scenarios that apply to SQL documents.

2. Set parameter values for SQL placeholders from the **Parameters** button from the scenario dialog. For example in `SELECT * FROM `test`.`department` where DEPT = ? or DEPTNAME = ?` two parameters can be configured for the place holders (?) in the transformation scenario.

When the SQL statement will be executed, the first placeholder will be replaced with the value set for the first parameter in the scenario, the second placeholder will be 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 is not recommended. It is due to a limitation of the SQL Server driver for Java applications. An example of stored procedure call that is not recommended is: `call dbo.Test(22, ?)`.

3. Execute the SQL scenario from the **Transform now** button of the scenario dialog.

The result of a SQL transformation will be *displayed in a view* at the bottom of the Oxygen XML Editor plugin 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 value or a CLOB one) 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 popup menu.
The action will copy the value in the clipboard.
- c) Paste the value where you need it.

For example you can paste the value in an opened XQuery editor panel of Oxygen XML Editor plugin.

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 plugin 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 plugin, go to http://www.oxygenxml.com/demo/Author_Database_XML_Native.html.

Configuring Database Data Sources

This section describes the procedures for configuring the data sources for native databases.

How to Configure a Berkeley DB XML Data Source

The latest instructions on how to configure Berkeley DB XML support in Oxygen XML Editor plugin can be found on our [website](#).

Oxygen XML Editor plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16.

1. Go to menu **Preferences > 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** combo box.
5. Press 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 into `DBXML_DIR / lib` or `DBXML_DIR / jar`)
- dbxml.jar (check for it into `DBXML_DIR / lib` or `DBXML_DIR / jar`)

Where `DBXML_DIR` is the Berkeley DB XML database root directory. For example on 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 an eXist Data Source

The latest instructions on how to configure eXist support in Oxygen XML Editor plugin can be found on our [website](#).

Oxygen XML Editor plugin supports eXist database server versions 1.3, 1.4 and 1.5.

1. Go to menu **Preferences > 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** combo box.
5. Press 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.

- exist.jar
- lib/core/xmlldb.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



Note: For eXist database server version 1.5 and 2.0, the following driver files must also be added in the dialog box for setting up the datasource:

- lib/core/slf4j-api-1.x.x.jar
- lib/core/slf4j-log4j12-1.x.x.jar


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.

How to Configure a MarkLogic Data Source

Available in the Enterprise edition only.

 **Note:** Oxygen XML Editor plugin supports MarkLogic version 4.0 or later.

The latest instructions on how to configure MarkLogic support in Oxygen XML Editor plugin can be found on our [website](#).

1. Go to menu **Preferences > 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 **Type** combo box.
5. Press 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 Documentum xDb (X-Hive/DB) 10 Data Source

Available in the Enterprise edition only.

The latest instructions on how to configure support for Documentum xDb (X-Hive/DB) 10 versions 8 and 9 in Oxygen XML Editor plugin can be found on our [website](#).

1. Go to menu **Preferences > 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** combo box.
5. Press 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.

Configuring Database Connections

This section describes the procedures for configuring the connections for native databases.

How to Configure a Berkeley DB XML Connection

Oxygen XML Editor plugin 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 the following:

1. Go to menu **Preferences > 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** combo box.
5. Fill-in 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 an eXist Connection

The steps for configuring a connection to an eXist database are the following:

1. Go to menu **Preferences > 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** combo box.
5. Fill-in 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.

The Create eXist-db XML connection Dialog

A quick way to create an eXist connection is to use the dedicated **Create eXist-db XML connection** dialog. Go to **Options > Preferences > Data Sources** and click **Create eXist-db XML connection**. After you fill in the fields in this dialog, click **OK** and go to **Window > Show View > Data Source Explorer** to view your connection.

To create an eXist connection using this dialog, Oxygen XML Editor plugin expects the `exist/webstart/exist.jnlp` path to be accessible at the **Host** and **Port** provided.

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.

 **Note:** Oxygen XML Editor plugin supports MarkLogic version 4.0 or later.

The steps for configuring a connection to a MarkLogic database are the following:

1. Go to menu **Preferences > 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** combo box.

5. Fill-in the connection details.

- a) The host name or IP address of the installed MarkLogic engine in the **XDBC Host** field.
Oxygen XML Editor plugin 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 on, 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. In case 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 an Documentum xDb (X-Hive/DB) 10 Connection

The steps for configuring a connection to a Documentum xDb (X-Hive/DB) 10 database are the following.



Note: The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Editor plugin. The following procedure explains the *xhive://* protocol connection type.

1. Go to menu **Preferences > 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** combo box.
5. Fill-in the connection details.
 - a) 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.
 - b) Set the user name to access the Documentum xDb (X-Hive/DB) 10 engine in the **User** field.
 - c) Set the password to access the Documentum xDb (X-Hive/DB) 10 engine in the **Password** field.
 - d) Set the name of the database to access from the Documentum xDb (X-Hive/DB) 10 engine in the **Database** field.
 - e) Check the checkbox **Run XQuery in read / write session (with committing)** if you want to end the session with a commit, otherwise the session ends with a rollback.
6. Click the **OK** button to finish the connection configuration.

Data Source Explorer View

This view presents in a tree-like fashion the database connections configured in the **Options > Preferences > Data Sources** preferences page. You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.

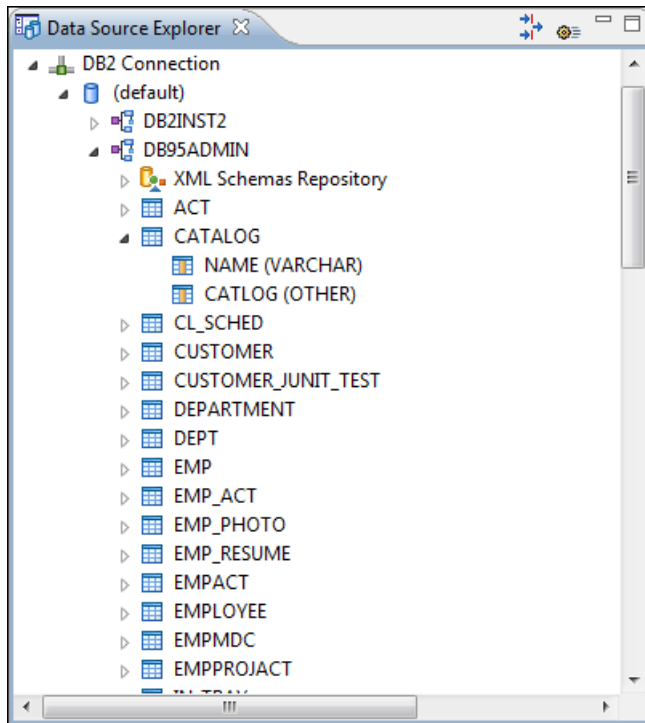


Figure 318: Data Source Explorer View

The following objects are displayed in the **Data Source Explorer** view:

- **Connection** ;
- **Catalog (Collection)**;
- **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 further sub-collections. There are two types of resources:

- **XML resource** - an XML document or a document fragment, selected by a previously executed XPath query.
- **non XML resource**

Note: For some connections you can add or move resources into a container by dragging them from:

- **Project view**;
- the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example);
- or from another database container.

The following actions are available in the view's toolbar:

- The **Filters** button opens the **Data Sources / Table Filters Preferences page**, allowing you to decide which table types will be displayed in the **Data Source Explorer** view.
- The **Configure Database Sources** button 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 plugin allows the user to browse the native Oracle XML Repository and perform various operations on the resources in the repository.

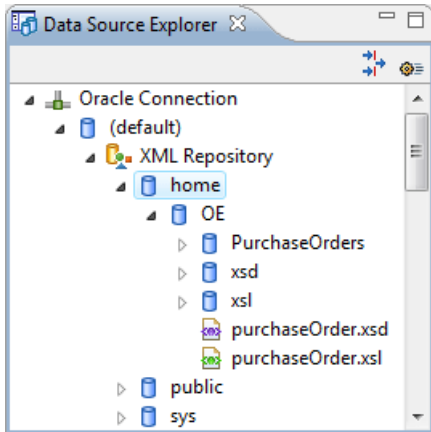






Figure 319: Browsing the Oracle XML DB Repository



The actions available at XML Repository level are the following:

-  **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 the following:

-  **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 the following:

-  **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 clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.
- **Properties** - Shows various properties of the current resource.
- **Compare** - This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

For running XQuery transformation on collections from XML Repository please see [a tutorial from Oracle](#).

PostgreSQL Connection

Oxygen XML Editor plugin allows the user to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.

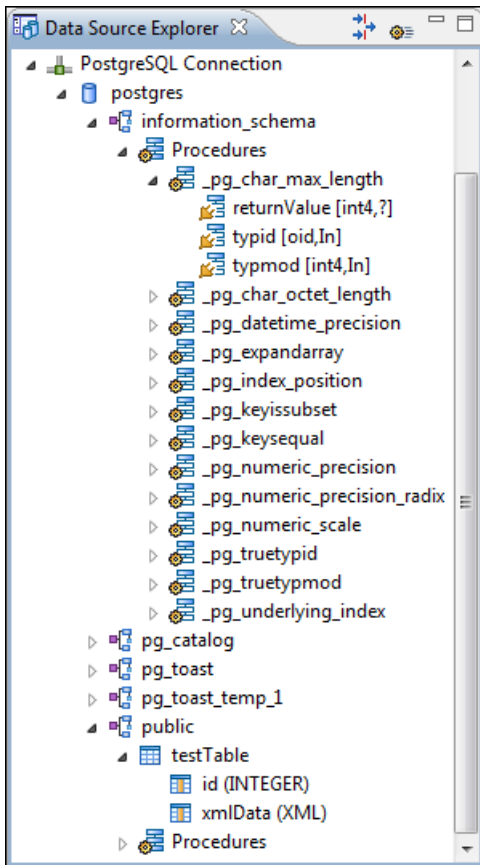






Figure 320: Browsing a PostgreSQL repository

The actions available at container level are the following:

-  **Refresh** - Performs a refresh of the selected container.

The actions available at resource level are the following:



-  **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** - This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

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 the following:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **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 on 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. That is, 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 needs 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 containing 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 the following:

-  **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 of the selected node's subtree.
- **Properties** - Displays a dialog with a list of properties of the Berkeley container like: container type, auto indexing, page size, validate on load, compression algorithm, number of documents, etc.

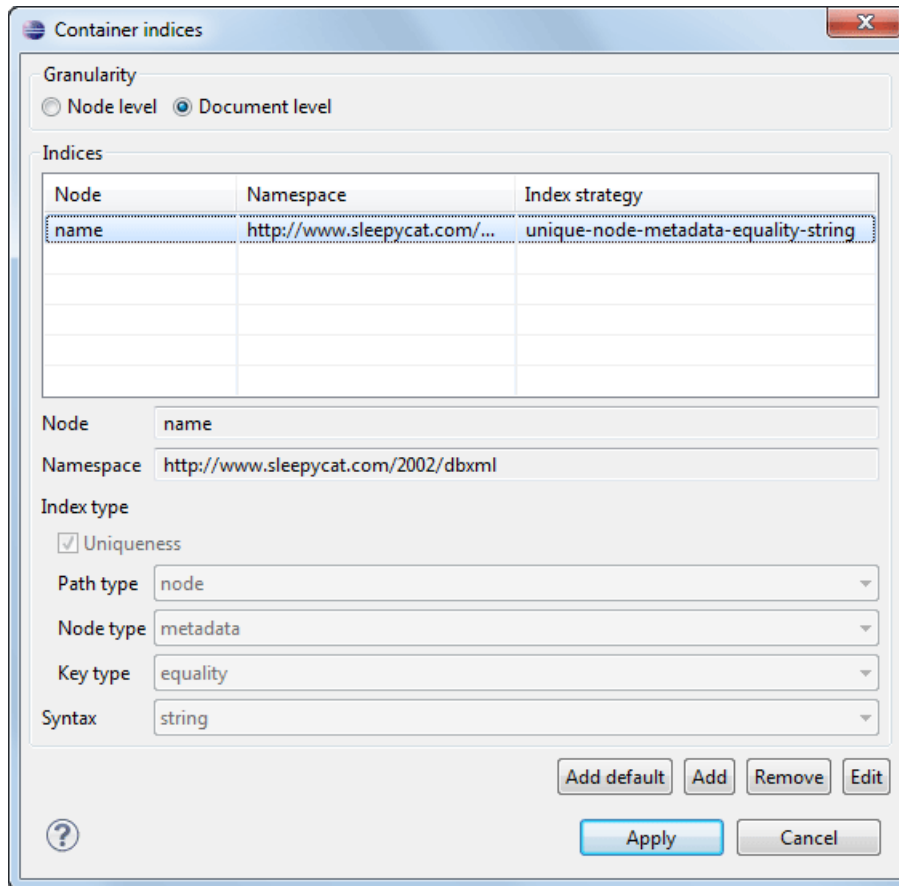


Figure 321: Container indices




The fields of the dialog are the following:

- Granularity:
 - **Document level** granularity is good for retrieving large documents.
 - **Node level** granularity is good 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 a document's metadata content.
 - **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 substring
- **Syntax types** - The syntax describes what sort of data the index will contain 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 the following:



-  **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 clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.
- **Compare** - This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

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 the following:

-  **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 of the selected node's subtree.


Actions Available at Container Level


For an eXist database the actions available at container level in the **Data Source Explorer** view are the following:

- **New File** - Creates a file in the selected container
- **New Collection** - Creates a collection
- **Import Folders** - Adds recursively the content of specified folders from the local filesystem
-  **Import Files** - Adds a set of XML resources from the local filesystem
- **Cut** - Cuts the selected containers
- **Copy** - Copies the selected containers







Note: You can add or move resources into container by dragging them from Project view, the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) or from another database container.

- **Paste** - Paste resources into 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 the user to view various useful properties associated with the container, like: name, creation date, owner, group, permissions.


Actions Available at Resource Level

For an eXist database the actions available at resource level in the **Data Source Explorer** view are the following:

-  **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 container by dragging them from Project view, the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) 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 clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.
 - **Properties** - Allows the user 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** - This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

MarkLogic Connection

Once you configure a MarkLogic connection, you can use the **Data Source Explorer** view to display all the application servers 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. In case this property is set to *manual* or *manual-enforced*, add the directory properties of the modules manually, using the XQuery function `xdrm: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:

```
(xdrm:directory-create('/code/modules/'),
xdrm:directory-create('/code/modules/imports/')).
```

For further information about directory properties go to: <http://blakeley.com/blogfile/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 plugin presents both the queries stopped for debugging and the queries that are still running. At the end of your session, to clean up the entire **Requests** container, right click it and use the **Cancel all running requests** action.

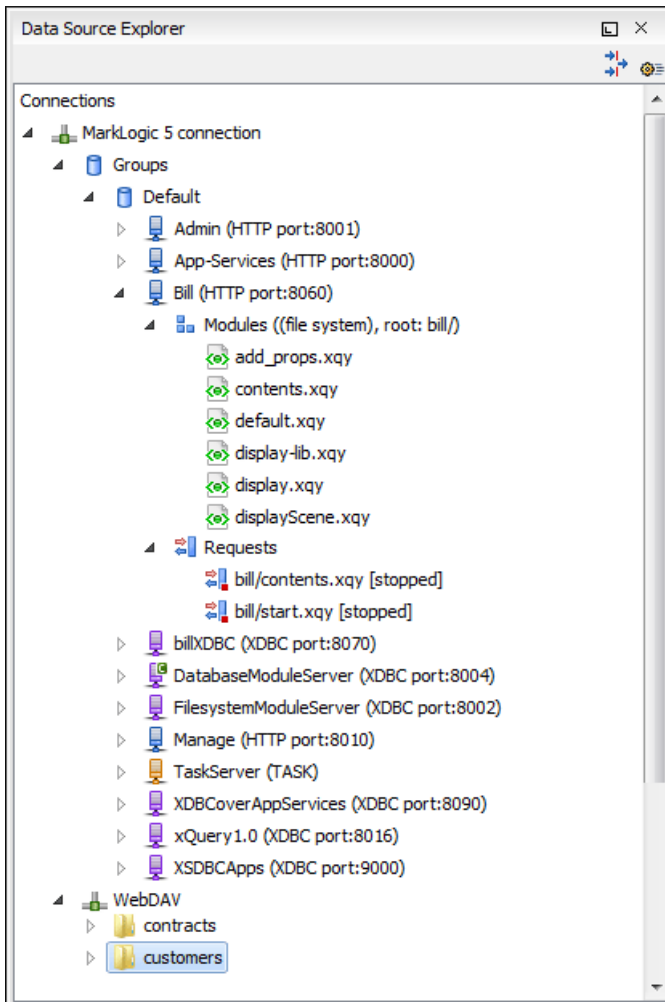















Figure 322: MarkLogic Connection in Data Source Explorer

The **Data Source Explorer** view presents all the application servers available on the MarkLogic server. To change the XDBC application server that Oxygen XML Editor plugin 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;
- **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;
 - **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;
 - **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 dialog;
 -  **Find/Replace in Files...** - Allows you to find and replace text in multiple files from the server;
 - **Compare** - This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

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 the following:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **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 into 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 the following:

-  **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 the following:

-  **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 to clipboard the URL of the selected schema resource.
- **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 the following:

-  **Refresh** - Performs a refresh of the selected library.
- **Add library** - Adds a new library as 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 would be 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 the following:

-  **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 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 to clipboard the URL of the selected resource.
- **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** - This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

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's or XML schemas are stored in the catalog during validated parsing.
 - **xhive-store-schema-only-internal-subset** - Stores only the internal subset of the document (not any external subset). This options modifies the **xhive-store-schema** one (only has a function when that parameter is set to true, and when DTD's are involved). Select this option this option if you only want to store the internal subset of the document (not the external subset).
 - **xhive-ignore-catalog** - Ignores the corresponding DTD's 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** - Convenience setting. With this 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-shrunked.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 which is useful for querying XML views of relational data to create XML results. It provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data as well. The following database systems supported in Oxygen XML Editor plugin 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 Data Source Explorer View


When a query is edited in the XQuery editor the XPath expressions can be composed quickly with drag and drop actions from the **Data Source Explorer** view to 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 up to the table or column that you want to insert in the query.
4. Drag the table name or the column name to the XQuery editor panel.
5. Drop the table name / column name where the XPath expression is needed.

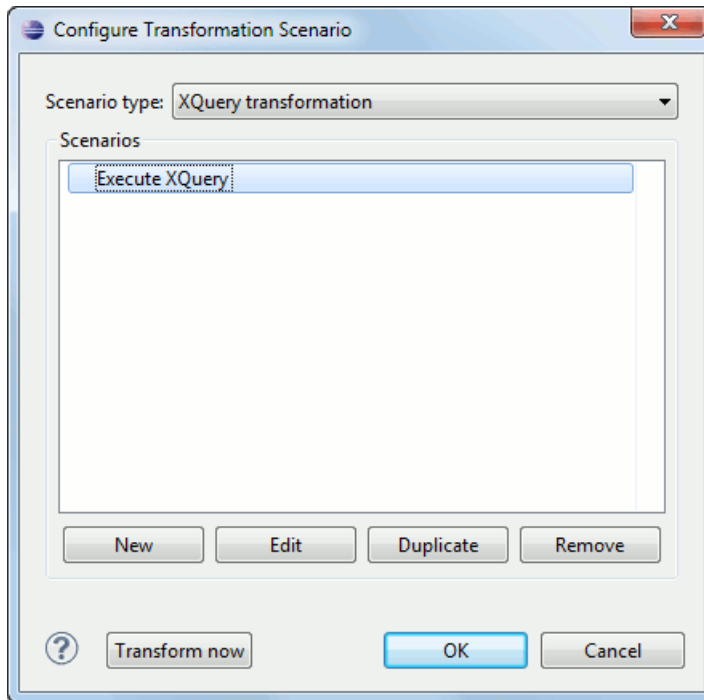
An XPath expression that selects the dragged name will be inserted in the XQuery document at caret position.

XQuery Transformation

XQuery is designed to retrieve and interpret XML data from any source, whether it is a database or a document. Data is stored in relational databases but often it is required that data is 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 dialog for configuring a scenario will be opened.



b) Click the **New** button of the dialog.

The dialog for editing an XQuery scenario will be opened.

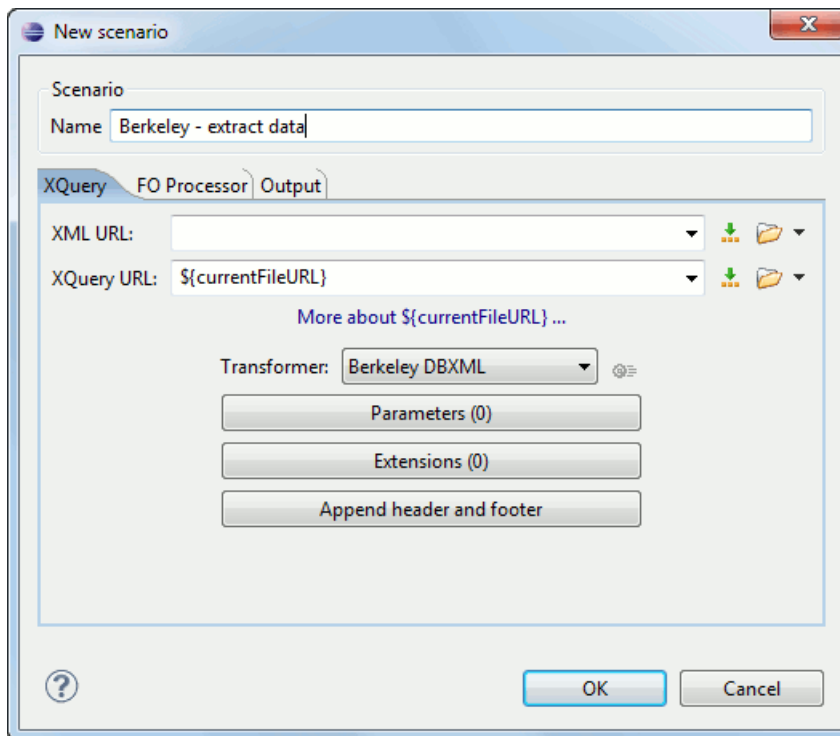


Figure 323: Edit Scenario Dialog

- c) Insert the scenario name in the dialog for editing the scenario.
- d) Choose the database connection in the **Transformer** combo box.
- e) Configure any other parameters if necessary.

For an XQuery transformation the output tab has an option called **Sequence** which allows you to execute an XQuery in lazy mode. The amount of data extracted from the database is controlled from 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 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 inserting in the query only calls to the supported functions (standard XQuery functions or extension ones).



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 displayed entirely in the XQuery query result table at the bottom of the Oxygen XML Editor plugin window, for example an XMLTYPE value or a CLOB value, do the following actions:

- right click on that table cell;
- select the **Copy cell** action from the popup menu for copying the value in the clipboard;
- paste the value where you need it, for example an opened XQuery editor panel of Oxygen XML Editor plugin.

XQuery Database Debugging

This section describes the procedures for debugging XQuery transformations that are executed against MarkLogic databases and Berkeley DB XML ones.

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 which Oxygen XML Editor plugin 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*). In case 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.

In case 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 resolving of any imported modules.

Before starting a debugging session, we recommend you to link the MarkLogic connection with an Eclipse project. To do this, go to the **Data Source Explorer** view and select **Link to project** in the contextual menu of the MarkLogic connection. The major benefit of linking a debugging session with a project is that you can add breakpoints in the XQuery modules stored on the server. You are also able to access these modules from the Eclipse navigator and run debugging sessions starting from them.

Oxygen XML Editor plugin supports collaborative debugging. This feature allows multiple users of Oxygen XML Editor plugin to participate in the same debugging session. You can start a debugging session and from 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 plugin 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 opened from the connection that participates at the debugging session.

To watch our video demonstration about the XQuery debugger for MarkLogic, go to <http://oxygenxml.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 which 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*;
- No support for *Output to Source Mapping*;
- 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 opened the modules in the Editor.
- Set no breakpoints in modules from the same server which 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 which import modules stored in the database the recommended steps for placing a breakpoint in a module are the following:

1. Start the debugging session with the action  **Debug Scenario** from the **Transformation** toolbar or the  **XQuery Debugger** toolbar button.
2.  **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 which you will debug are already opened in the editor. This is necessary so that the breakpoints in modules will be considered. Also make sure there are no other opened modules which 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 2.5.13 and it is supported in Oxygen XML Editor plugin's XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic one.

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 plugin 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 will be available in *the Data Source Explorer view*. The steps for configuring a WebDAV connection are the following:

1. Go to menu **Preferences > 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** combo box.
5. Fill-in the connection details:
 - a) Set the URL to the WebDAV repository in the field **WebDAV URL**.
 - b) Set the user name to access the WebDAV repository in the field **User**.
 - c) Set the password to access the WebDAV repository in the field **Password**.
6. Click the **OK** button.





To watch our video demonstration about the WebDAV support in Oxygen XML Editor plugin, go to http://www.oxygenxml.com/demo/WebDAV_Support.html.

WebDAV Connection Actions

This section explains the actions that are available on 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;
-  **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;
- **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;
- **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 dialog;
-  **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 an 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. The configuration file for the HTTP server is named `.basex` and is located in the BaseX installation directory. This file might help you to find out the port on which the HTTP server is running. The default port for BaseX WebDAV is 8984.

To ensure everything is functioning, open an WebDAV URL inside a browser and check if it works. For example, the following URL gets 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 plugin as described in [How to Configure a WebDAV Connection](#) on page 664. 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 596. 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. As an alternative, you can also find the libraries in the BaseX installation directory, in the `lib` subdirectory.

BaseX XQJ Connection

The next step is to create an XQJ connection as described in [How to Configure an XQJ Connection](#) on page 596.

For a default BaseX configuration, the following connection details apply (please modify them when necessary):

- *Port*: 1984
- *serverName*: localhost
- *user*: admin
- *password*: admin

XQuery execution

Now that the XQJ connection is configured, open in Oxygen XML Editor plugin the XQuery file you wish to execute and create a *Transformation Scenario* as described in [XQuery Transformation](#) on page 566. In the **Transformer** combo box, select the name of the XQJ connection you created. Apply the transformation scenario and the XQuery will be executed.

Chapter 16

Importing Data

Topics:

- [Introduction](#)
- [Import from Database](#)
- [Import from MS Excel Files](#)
- [Import from HTML Files](#)
- [Import from Text Files](#)

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 plugin 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*.

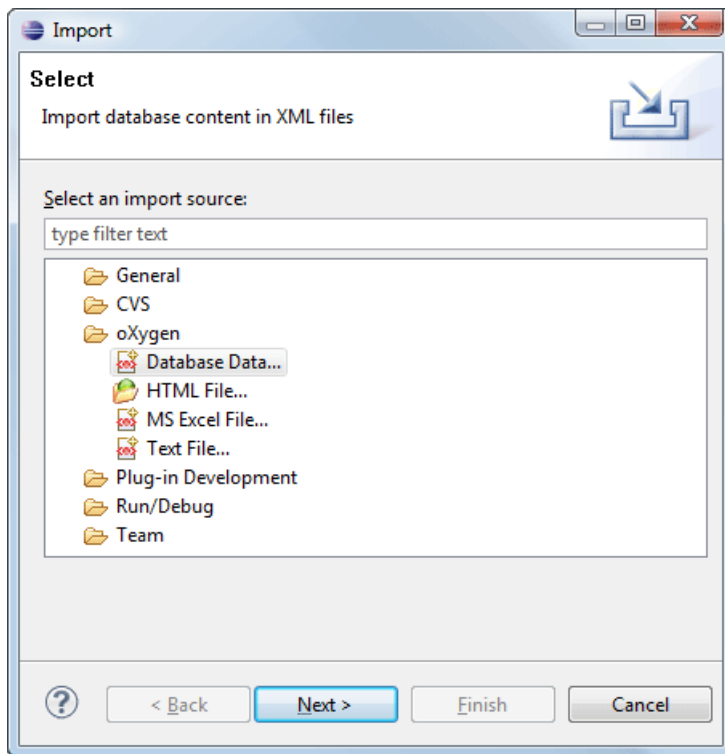


Figure 324: The Import Wizards of the Oxygen XML Editor plugin Plugin

Import from Database

This section explains how to import data from a database into Oxygen XML Editor plugin.

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 > oXygen / Database Data**.

Clicking this action opens a dialog with all the defined database connections:

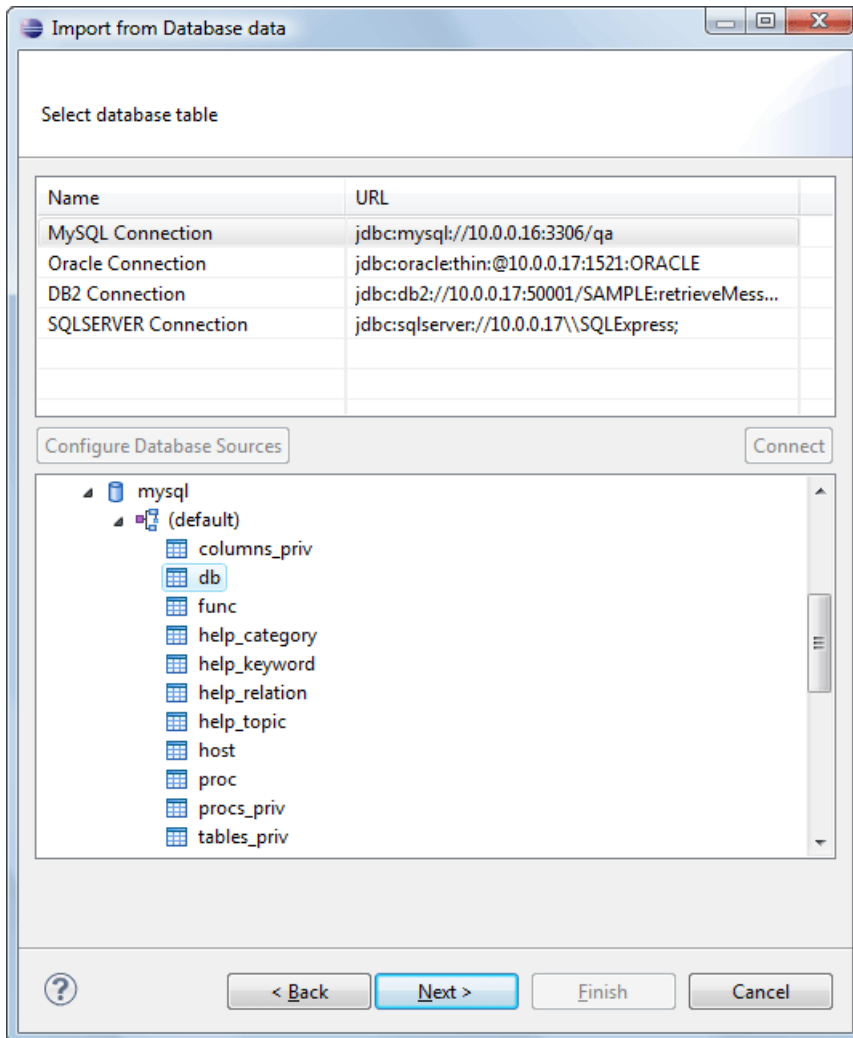


Figure 325: Import From Database Data Wizard

2. Select the connection to the database that contains the data.
Only connections configured on 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 opens next, with a default query string in the **SQL Query** pane:

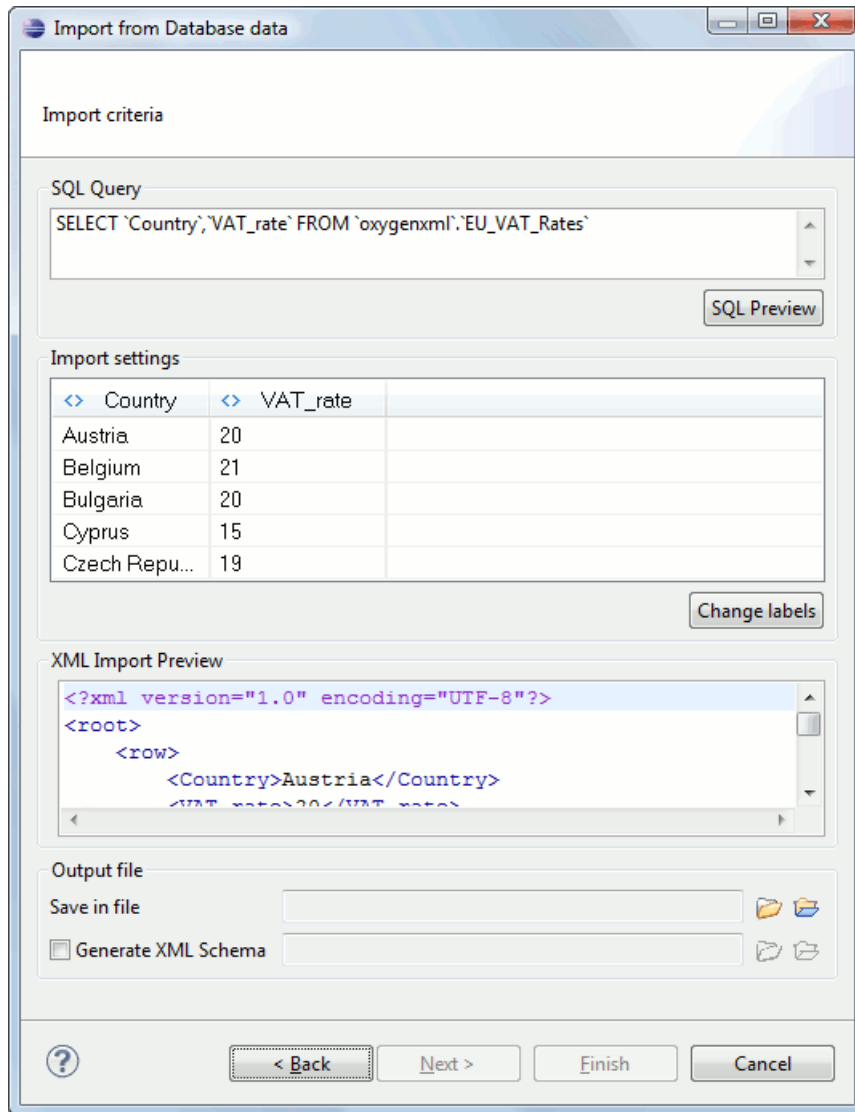


Figure 326: Import from Database Criteria Dialog

The dialog contains the following items:

- **SQL Preview** - If the **SQL Preview** button is pressed, it shows the labels that are used in the XML document and the first five lines from the database into the **Import settings** panel. 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 on a column header (ex **Heading0**) causes the data from this column to be used as attribute values of the row elements. Click a 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
- **Change labels** - This button opens a new dialog, allowing 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 from the drop-down list **ELEMENT**, **ATTRIBUTE**, or **SKIPPED**.
- **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 open in the editor, but 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 are true:

- the query string represents a join operation of two or more tables
- columns 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 a tabular form in the **Import Settings** panel. The **XML Import Preview** panel 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 plugin as an XML Schema. This feature is activated by the **Generate XML Schema** option from the **Import criteria** dialog used in [the procedure for importing table data](#) as an XML instance document.

Import from MS Excel Files

Oxygen XML Editor plugin 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, 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.

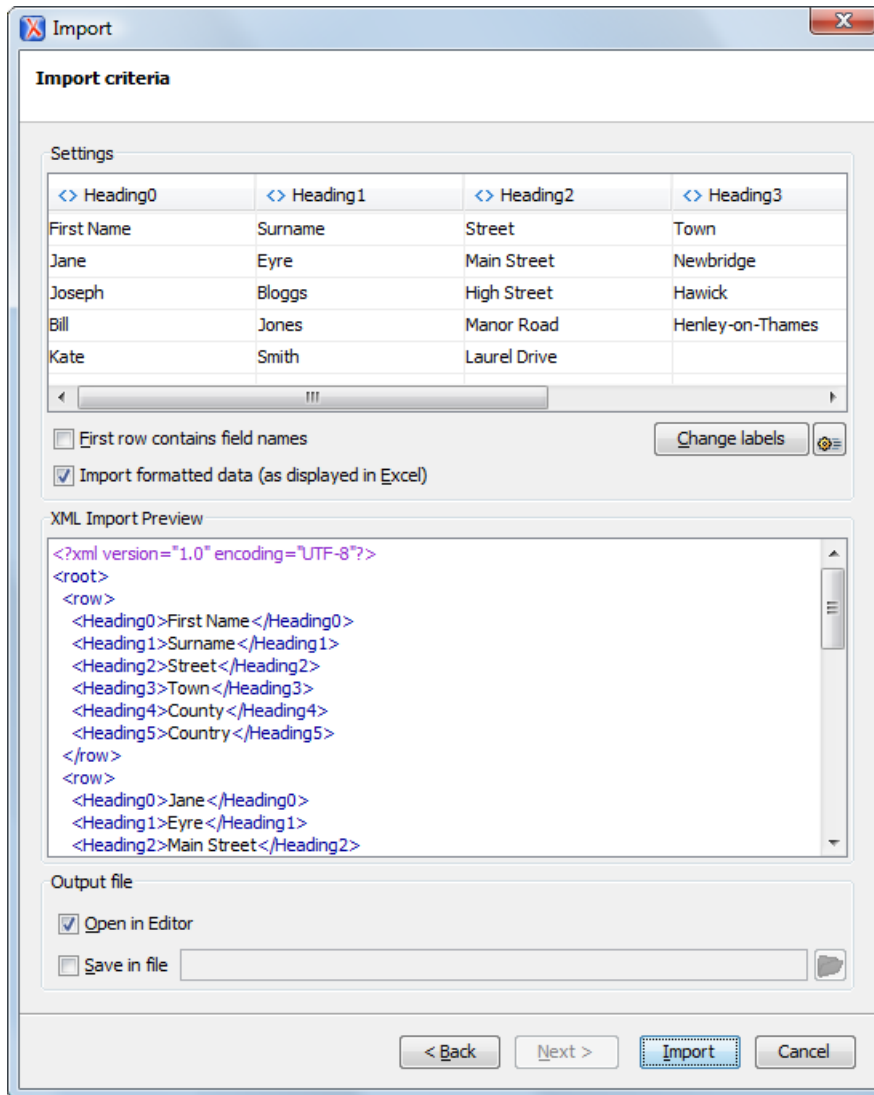


Figure 327: The "Import" Dialog Box - Import Criteria

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;

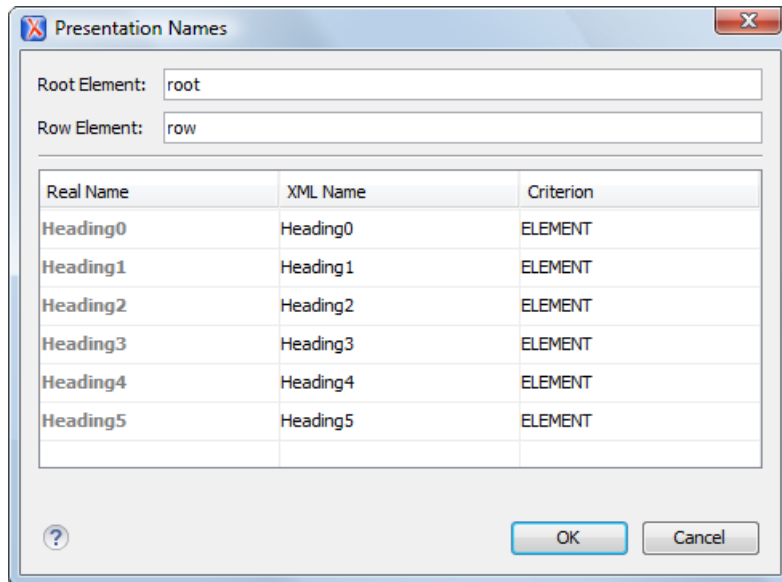


Figure 328: The "Presentation Names" Dialog Box

The following options are available in this dialog:

- **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 dialogs, click **Import**.

Import from MS Excel 2007-2010 (.xlsx)

You need to add additional JAR libraries to Oxygen XML Editor plugin To import XML from Excel 2007-2010 (.xlsx) documents.

First you need to download the latest stable release of the Apache POI project from <http://poi.apache.org/download.html>.

From the downloaded project locate and add the following .jar files in the plugin.xml file:

- dom4j-1.6.1.jar;
- poi-ooxml-3.8-20120326.jar;
- poi-ooxml-schemas-3.8-20120326.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 > oXygen > HTML File ...**
The **Import HTML** wizard 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 resulted document is an XHTML file containing a DOCTYPE declaration referring to 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 > oXygen > 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:

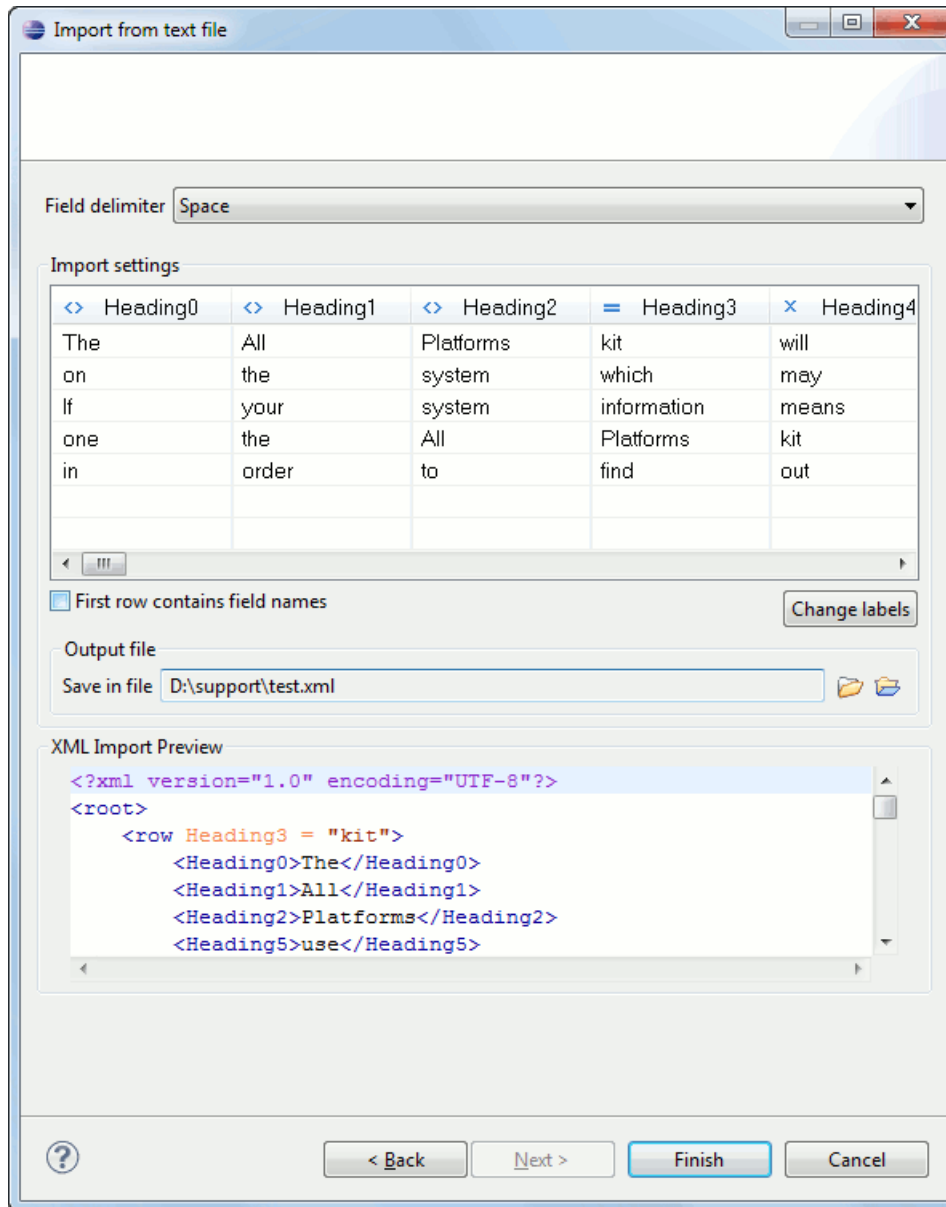


Figure 329: Import from text file

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 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, allowing 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**.

- **Output file** - Allows you to select the output XML file.

Chapter 17

Content Management System (CMS) Integration

Topics:

- [Integration with Documentum \(CMS\)](#)
- [Integration with Microsoft SharePoint](#)

This chapter explains how you can integrate Oxygen XML Editor plugin with a content management system (CMS), to edit the data stored in the CMS directly in Oxygen XML Editor plugin. .

Integration with Documentum (CMS)

Oxygen XML Editor plugin 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 plugin supports Documentum (CMS) version 6.5 or later with *Documentum Foundation Services 6.5* or later 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. Go to menu **Preferences > Data Sources**.
The **Preferences** dialog is opened at the **Data Sources** panel.
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. Go to menu **Preferences > 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 and later versions do not exhibit this problem.



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 plugin 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 plugin 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.

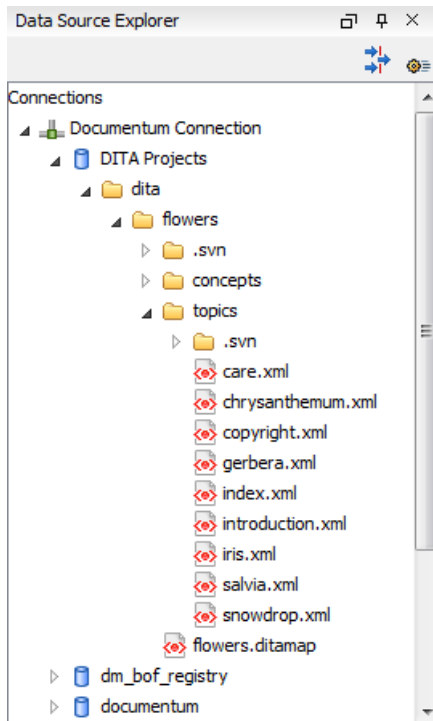




Figure 330: Browsing a Documentum repository



Actions Available on Connection

The actions available on a Documentum (CMS) connection in the **Data Source Explorer** view are the following:

-  **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 in the import dialog:
 - **Add Files** - Shows a file browse dialog and allows you to select files to add to the list.
 - **Add Folders** - Shows a folder browse dialog that allows you to select folders to add to the list. The subfolders will be added recursively.
 - **Edit** - Shows a dialog 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 subtree.
- **📄 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.
- **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.

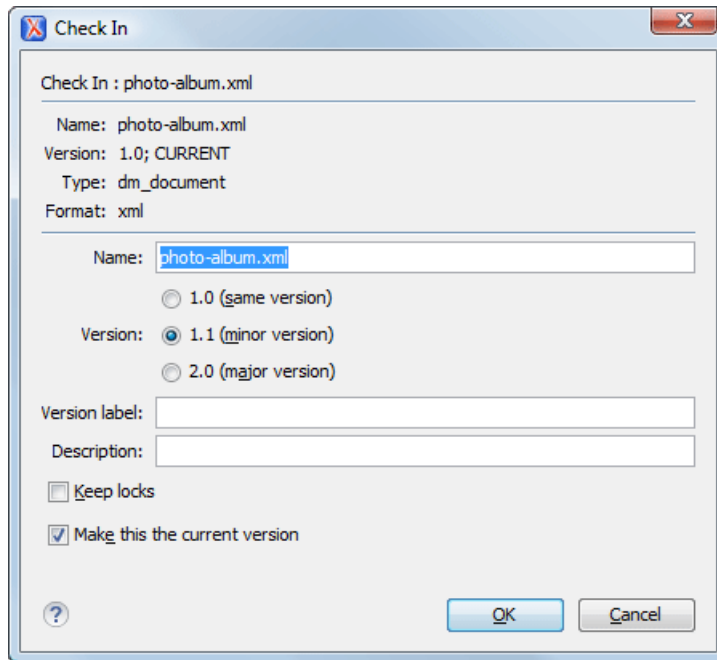



Figure 331: Check In Dialog

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 Mac OS))** 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 plugin 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 plugin 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 and SharePoint Online for Office 365

How to Configure a SharePoint Connection

By default Oxygen XML Editor plugin is configured to contain a SharePoint data source connection called **SharePoint**. Based on this data source you create a SharePoint connection to a SharePoint server. The connection will be available in [the Data Source Explorer view](#). The steps for configuring a SharePoint connection are the following:

1. Go to menu **Preferences > 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.




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;
- **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 dialog;
-  **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 18

Tools

Topics:

- [XML Digital Signatures](#)

Oxygen XML Editor plugin ships with a set of tools oriented to XML related tasks. You gave 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.

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 plugin: 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.

These three operations: Digital Signing, Canonicalization and Verification of the signature are available from the **Tools** menu or from the Editor's **contextual menu** > **Source**.

Canonicalizing Files

The user can select the canonicalization algorithm to be used for his document from the following dialog displayed by the action **Canonicalize** available from the editor panel's **contextual menu** > **Source** .

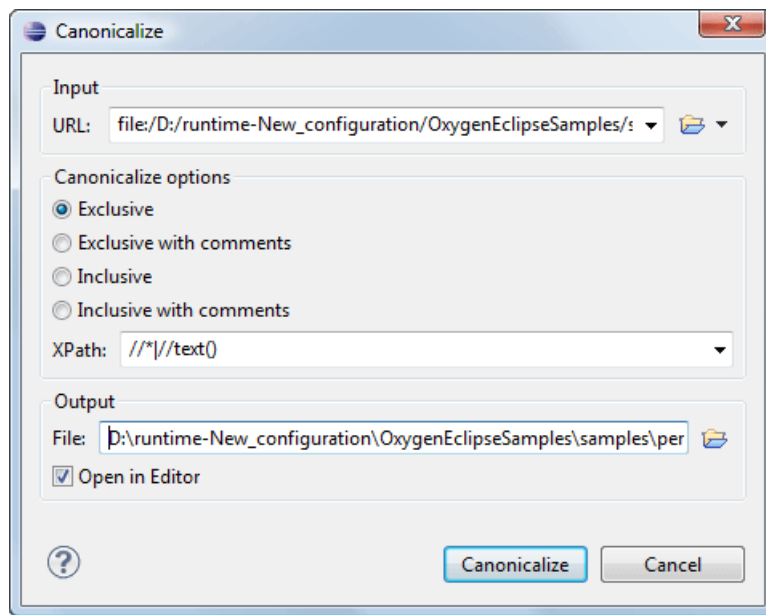


Figure 332: Canonicalization settings dialog

The fields of the dialog are 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 no keystore can store trusted certificates generated with keys in its keystore.

In Oxygen XML Editor plugin 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 set the options for a certificate or to validate it, go to menu *Options > Preferences > Certificates* .

Signing Files

The user can select the type of signature to be used for his document from the following dialog displayed by the action **Sign** available from the editor panel's **contextual menu > Source**

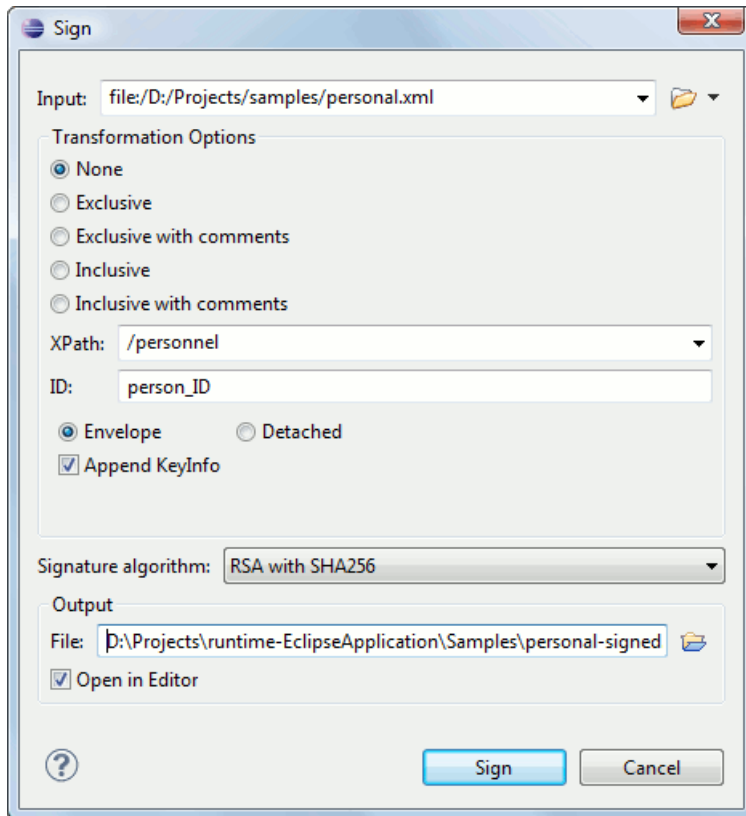


Figure 333: Signature settings dialog

The following options are available:

- **Input** - Specifies the location of the input URL.
- **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 enveloping signature is used.
- **Detached** - If selected, the detached signature is used.
- **Append KeyInfo** - The element `ds:KeyInfo` will be added in the signed document only if this option is checked.
- **Signature algorithm** - 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 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.

Verifying the Signature

You can verify the signature of a file from the contextual menu of the file: **Source > Verify Signature**. The **URL** field in the **Verify Signature** dialog specifies the location of the file whose signature is verified.

In case the signature is valid, a dialog displaying the name of the signer is displayed. If not, an error shows details about the problem.

Chapter 19

Configuring Oxygen XML Editor plugin

Topics:

- [Configuring Options](#)
- [Importing / Exporting Global Options](#)
- [Preferences](#)
- [Reset Global Options](#)
- [Scenarios Management](#)
- [Editor Variables](#)
- [Localization of the User Interface](#)

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.

Configuring Options

A set of options controls Oxygen XML Editor plugin, allowing you to configure most of the features. To offer you the highest degree of flexibility in customizing the application to fit the end-user's role in your organization, Oxygen XML Editor plugin comes with several distinct layers of option values, arranged here according to their priority, from low to high:

- *Default Options* - should be regarded as factory defaults or built-in values.

This predefined set of values are tuned so Oxygen XML Editor plugin behaves optimally in most working environments.

- *Customized Default Options* - designed to customize the initial option values for a group of users.

This layer allows an administrator to deploy the application preconfigured with a standardized set of option values.



Note: Once this layer is set, it represents the initial state of Oxygen XML Editor plugin when an end-user presses the **Restore defaults** or **Reset Global Options** buttons.

- *Global Options* - allows individual users to personalize Oxygen XML Editor plugin according to their specific needs.



Note: If you set a specific option in one of the layers, but it is not applied in the application, make sure that one of the higher priority layers is not overwriting it.

Customized Default Options

You can overwrite the *Default Options* values by loading a set of options, from a specific XML structure, stored in an *options file* (a file that contains all the Oxygen XML Editor plugin options). To create an *options file*:

- set the values you want to impose as defaults in the *Preferences pages*;
- select **Window > Preferences > oXygen XML Editor > Export Global Options**.

There are two ways to make Oxygen XML Editor plugin use such an options configuration file to extract the customized default options:

- set the file path of the options configuration file as value of `com.oxygenxml.default.options` Java virtual machine parameter.

The file must be specified with an URL or a file path relative to the application installation folder. For example, add the following line in the `[Eclipse-platform-install-folder]/configuration/config.ini` file:

```
com.oxygenxml.default.options=@config.dir/../../default-options.xml
```

- copy the options configuration file in the `[Eclipse-platform-install-folder]/plugins/com.oxygenxml.editor/preferences` folder.



Note: Make sure that the options configuration file has an `.xml` or an `.xpr` extension, like `default-options.xml`.

Importing / Exporting Global Options

The import/export buttons are located in the preferences page of the Oxygen XML Editor plugin. To open this page, go to **Window > Preferences > oXygen XML Editor**. You can use the import/export buttons to load or save global preferences as an XML file which can be reloaded both on your computer and on others.


The following actions are available in the **Options** menu:

- **Reset Global Options** - restores the preferences set to *Customized Default Options layer*, or if this level is not defined, to *Default Options layer*;
- **Import Global Options** - allows you to import a set of *Global Options* from an *options file*;
- **Export Global Options** - allows you to export the entire set of *Global Options* to an options file.

Preferences

Once the application is installed you can use the **Preferences** dialog accessed from menu **Options > Preferences** to customize the application.

You can always revert modifications to their default values by pressing the **Restore Defaults** button, available in each preferences page.

If you don't know how to use a specific preference that is available in any **Preferences** panel or what effect it will have you can open a help page about the current panel at any time pressing the help button  located in the left bottom corner of the dialog or pressing the F1 key.

A restricted version of the **Preferences** dialog is available at any time in the editors of the Oxygen XML Editor plugin by right-clicking in the editor panel and selecting **Preferences**:

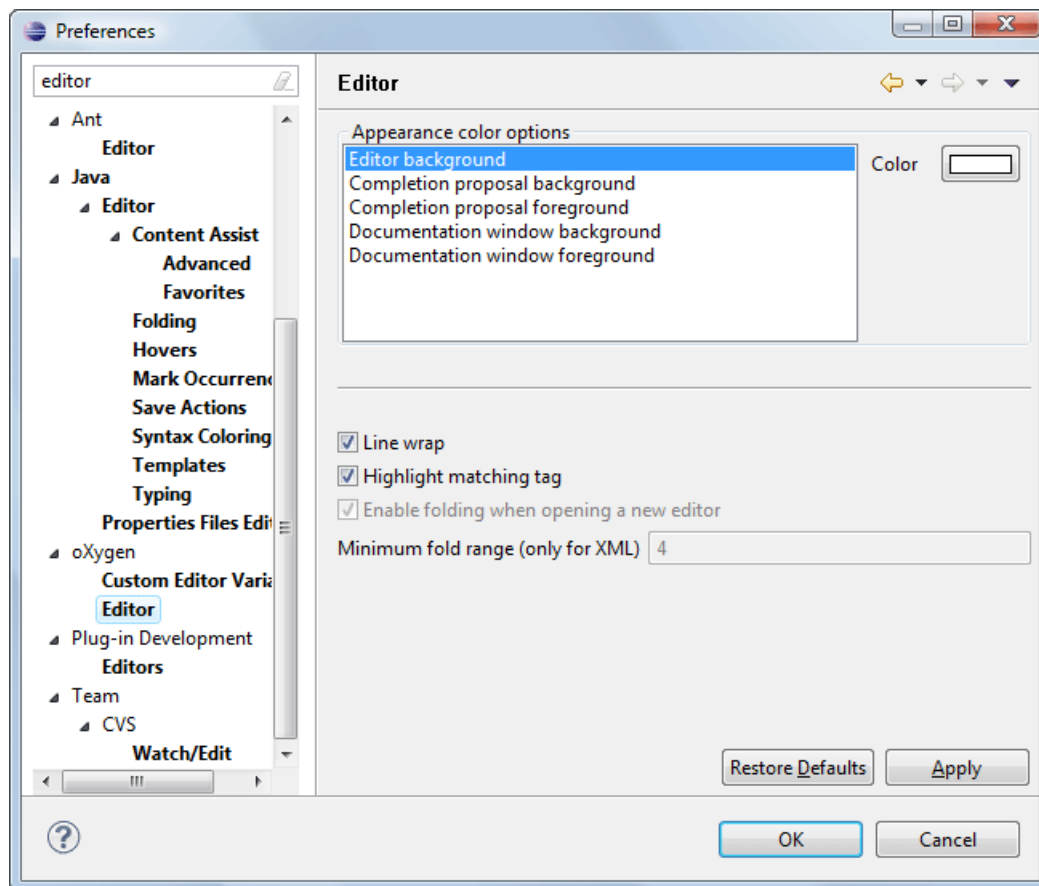


Figure 334: Eclipse Preferences dialog - restricted version

Oxygen XML Editor plugin License

To view the license preferences, go to **Window > Preferences > oXygen XML Editor**. This preferences page presents the details of the license key which enables the Oxygen XML Editor plugin: registration name, category and number of purchased licenses, encrypted signature of the license key. Clicking the **Register** button to open the Oxygen XML Editor plugin **License** dialog, which allows you to insert a new license key.

Fonts Preferences

To open the **Fonts** preferences page, go to **Window > Preferences > oXygen XML Editor > Fonts**.

The following options are available:

- **Text** - The family and size of the font used in text-based editors. There are two options available:
 - **Map to text font** - Uses the same font as the one set in **General / Appearance / Colors and Fonts** for the basic text editor.
 - **Customize** - Allows you to choose a specific font.
- **Author** - Allows you to specify a font to be used in **Author** mode.

To edit the fields use the **Change** and **Reset** buttons available in each section of the **Fonts** preferences page. The **Change** button opens a dialog that allows you to configure the **Font family**, **Font size**, **Font style**, **Effects**, **Color** and **Script**. A preview window is also available. The **Reset** button restores the setting to their default values.

Document Type Association Preferences

A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the **Author** mode for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both **Author** mode and Text mode;
- CSS stylesheet(s) for rendering XML documents in **Author** mode;
- user actions invoked from toolbar or menu in **Author** mode;
- predefined scenarios used for transformation of 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.

To open the **Document Type Association** preferences page, go to **Window > Preferences > oXygen XML Editor > Document Type Association**.

Oxygen XML Editor plugin is highly customizable. Practically you can associate an entire class of documents (grouped logically by some common features like namespace, root element name or filename) to a bundle consisting of CSS stylesheets, validation schemas, catalog files, new files templates, transformation scenarios and even custom actions. The bundle is called *document type* and the association is called *Document Type Association* or, more generically, *framework*.

The following actions are available in this preferences panel:

- **Change framework directory location** - specifies a custom frameworks folder from where Oxygen XML Editor plugin loads the document types;
- **Document types table** - presents the currently defined document type associations, ordered by priority and alphabetically. Each row of the table represents a document type association, each holding the following information:
 - **Document type** - name of the document type;
 - **Enabled** - when checked, the corresponding document type association is enabled and analyzed when the application is trying to determine the type of an opened document;

- **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: Note that 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 application's internal options and the file is removed.

- **Priority** - depending on the priority level, Oxygen XML Editor plugin 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, Highest. You can set a higher priority to Document Type Associations you want to be evaluated first;

When expanding a **Document Type Association** its defined rules are presented. 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 is in 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 allowing you to edit an existing association;



Note: If you try to edit an existing association type when you have no write permissions to its store location, a dialog box will be shown, asking if you want to duplicate the 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 plugin 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

The **Locations** preferences page allows you to specify the location from which the frameworks used in Oxygen XML Editor plugin are loaded. You can choose between `OXYGEN_INSTALLATION_FOLDER/frameworks`, or a custom directory. You are also allowed to specify additional frameworks directories. The frameworks are loaded from the current framework directory, from add-ons, and from the additional frameworks directories. The priority of the frameworks is internal frameworks, followed by additional ones and external frameworks.

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*;
- **Priority** - depending on the priority level, Oxygen XML Editor plugin 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, Highest. You can set a higher priority to Document Type Associations you want to be evaluated first;
- **Description** - a detailed description of the framework;
- **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: Note that 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 application's internal options and the file is removed.

- **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 plugin to identify the type of a document. An Oxygen XML Editor plugin *association rule* holds information about *Namespace*, *Root local name*, *File name*, *Public ID*, *Attribute*, and *Java class*. Oxygen XML Editor plugin identifies the type of a document when the document matches at least one of the *association rules*. Using the **Document type rule** dialog, you can create *association rules* that activate on any document matching all the criteria in the dialog.

In the **Association rules** tab you can perform the following actions:

- **New** - opens the **document type rule** dialog allowing you to create *association rules*;
- **Edit** - opens the **document type rule** dialog 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 plugin 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 refer 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 plugin 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 that allows you to add a resource in the **Classpath** tab;
-  **Edit** - opens a dialog 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 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 menu, toolbar 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 plugin uses to render a document in the **Author** mode. In this sub-tab, you set alternate CSS files. When you are editing a document in the **Author** mode, you are able to switch between these CSS files from the **Author CSS Alternatives** 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;
- **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 a 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

The **Action** dialog allows you to edit an existing document type action or create one. To open this dialog, go to **Option > Preferences > Document Type Association**, select a document type, and click **Edit** or **New**. The **Document type** dialog is presented. In this dialog go to the **Author** tab, click **Actions**, select an action, and click  **Edit** or  **New**.

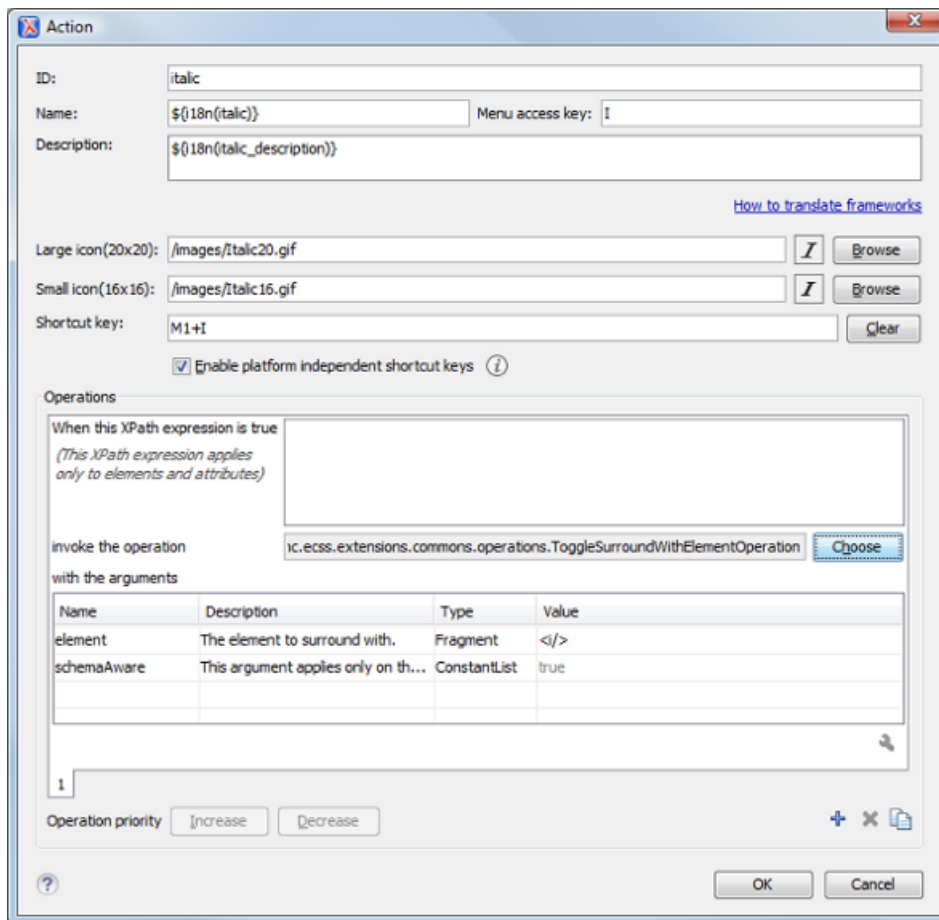


Figure 335: The Action Dialog

The following options are available:

- **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** - on 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 letter from the menu item name having the same value;
- **Description** - a description of the action;
- **Large icon** - select an image that Oxygen XML Editor plugin uses for the toolbar action.



Note: 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. In case the images are bundled in a *jar* archive, together with some Java operations implementation for instance, it is convenient to refer the images by their relative path location in the *class-path*.

- **Small icon** - select an image that Oxygen XML Editor plugin uses for the contextual menu icon;
- **Shortcut key** - this field allows you to configure a shortcut key for the action you are editing. The + character separates the keys. The shortcut you are specifying in this field is platform-independent. 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 you are configuring the functionality of the action 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.

The following options are available in this section:

- **When this XPath expression is true** - an XPath expression that applies to elements and attributes;



Note: Oxygen XML Editor plugin provides two XPath extension functions: *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.

- **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. In case 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.

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, $defaultAttributeValue, $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`.

defaultAttributeValue

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 `defaultAttributeValue` 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 plugin 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 plugin 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 plugin 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 plugin 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** dialog wizard.

The Catalogs Tab








The **Catalogs** tab specifies a list of *XML catalogs* which are added to all the catalogs that Oxygen XML Editor plugin 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 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 and are also displayed on the tooltip of the **Apply transformation Scenario(s)**.

The **Transformation** tab offers the following options:








-  **New** - opens the **document type rule** dialog allowing you to create *association rules*;
-  **Edit** - opens the **document type rule** dialog allowing you to edit the properties of the currently selected *association rule*;
-  **Delete** - deletes the currently selected *association rules*;
-  **Import scenarios** - imports transformation scenarios;
-  **Export selected scenarios** - export transformation scenarios;
-  **Move Up** - moves the selection to the previous *association rule*;
-  **Move Down** - moves the selection to the following *association rule*.

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 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 and are also displayed on the tooltip of the **Apply transformation Scenario(s)** button.

The **Validation** tab offers the following options:

-  **New** - opens the **document type rule** dialog allowing you to create *association rules*;
-  **Edit** - opens the **document type rule** dialog allowing you to edit the properties of the currently selected *association rule*;
-  **Delete** - deletes the currently selected *association rules*;
-  **Import scenarios** - imports transformation scenarios;
-  **Export selected scenarios** - export transformation scenarios;
-  **Move Up** - moves the selection to the previous *association rule*;
-  **Move Down** - moves the selection to the following *association rule*.

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.

Document Type Sharing

Oxygen XML Editor plugin 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_INSTALL_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_INSTALL_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_INSTALL_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_INSTALL_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_INSTALL_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_INSTALL_DIR/frameworks` directory. The new document type will be available in the list of Document Types when Oxygen XML Editor plugin 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.

Deploying Plugins or Frameworks as Add-ons

To deploy a plugin or a framework as an Oxygen XML Editor plugin add-on:

- Pack it as a ZIP file or a *JAR*. Please note that you should pack the entire root directory not just its contents;
- 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_install_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.

- Create a descriptor file. You can use a template that Oxygen XML Editor plugin provides. To use this template, go to **File > New** and select the **Oxygen add-ons update site** template;
- Copy the ZIP file and the descriptor file to an HTTP server. The URL to this location serves as the **Update Site URL**.

Localizing Frameworks

Oxygen XML Editor plugin 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 plugin (in **Options > Preferences > Global > Language** 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 plugin matches the GUI language with the language set in the `translation.xml` file. In case 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:

- framework actions;
- menu entries;
- contextual menus;
- toolbar;
- static CSS content.

For example, if you want to localize the bold action go to **Options > Preferences > Document Type Association**.

Open the **Document type** dialog, 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 for a certain key the associated value. In this way all the dialogs that you present from your custom operations can have labels translated in different languages.

You can also refer 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_INSTALL_DIR]/frameworks/docbook/i18n/translation.xml`. In the **Classpath** list corresponding to the Docbook document type the following entry was added: `#{framework}/i18n/`.

In **Options > Preferences > Document Type Association > Author > Actions**, you can see how the DocBook actions are defined to use these keys for their name and description. If you look in the Java class `ro.sync.ecss.extensions.docbook.table.SADocbookTableCustomizerDialog` available in the Author SDK, you can see how the new `ro.sync.ecss.extensions.api.AuthorResourceBundle` API is used to retrieve localized descriptions for different keys.

Editor Preferences

To open the **Editor** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor**.

Use these options to configure the visual aspect of the text editor. The same options panel is available in *the restricted version of the Preferences dialog*.

The following options are available:

- **Editor background** - Background color for both text editor and Diff Files editors.
- **Completion proposal background** - Background color of the content completion assistant window.
- **Completion proposal foreground** - Foreground color of the content completion assistant window.
- **Documentation window background** - Background color of the window containing documentation for the elements suggested by the content completion assistant.
- **Documentation window foreground** - Foreground color for the window containing documentation for the elements suggested by the content completion assistant.
-
- **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.
- **Highlight matching tag** - If you place the cursor on a start or end tag, Oxygen XML Editor plugin highlights the corresponding member of the pair. You can also customize the highlight color.
- **Display quick assist notification icon** - Displays the *Quick Fix* yellow bulb icon in the editor line number stripe.
- **Beep on operation finished** - Oxygen XML Editor plugin emits a short beep when a validate, check well-formedness, or transform action has ended;



Note: When the validation or the transformation process of a document is successful, the beep signal has a higher audio frequency, as opposed to when the validation fails, and the beep signal has a lower audio frequency. On the Windows platform, for other operations, the default system sound (*Asterisk*) is used. You can configure it by changing the sound theme.

- **Enable folding when opening a new editor** - Enables *folding support* when you open a new editor pane.
- **Minimum fold range (only for XML)** - If **Enable folding when opening a new editor** option is checked, you can specify the minimum number of lines in a block for which the folding support becomes active. If you modify this value, the change takes effect next time you open / reopen the editor.

Print Preferences

To open the **Print** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes**. This page allows you to customize the information printed on the header and footer of a page. These settings do not apply to the **Grid** and **Schema Diagram** modes.

You can customize the information printed in the one-row header and footer of a page. The following options are available:

- **Left, Middle** and **Right** area of the header and footer. Here you can write a pattern of the text printed in the header and footer of the page. You can use the following editor variables to write the text pattern:
 - **\${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 *cfne* 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>.



Note: As an example, to show the current page number against 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: ``${cp}` from `${tp}``.

- **Color** - Text color.
- **Font** - Font type. Default is `SansSerif`.
- **Underline/Overline** - When selected, a separator line is drawn between the header/footer and the page content.

Edit modes Preferences

The **Edit modes** preferences page allows you to select the initial edit mode of an editor. The previous editing mode is saved and used at start-up. To open the **Edit modes** preferences page, go to **Window > Preferences > oxygen XML Editor > Editor > Edit modes**.

If the checkbox **Allow Document Type specific edit mode setting to override the general mode setting** is enabled, the initial edit mode setting set in *the Document Type dialog* overrides the general edit mode setting from the table below.

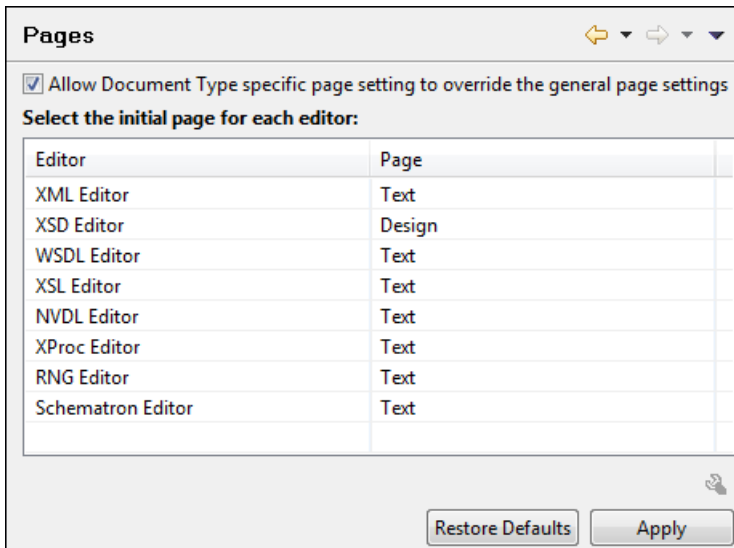
The initial edit mode of each editor type has one of the following values:

- Text;
- Author;
- Grid;
- Design (available only for the W3C XML Schema editor).

The Oxygen XML Editor plugin “Edit modes” Preferences Page

Editor / Edit modes	
<input checked="" type="checkbox"/> Allow Document Type specific edit mode setting to override the general edit mode settings	
Select the initial edit mode (page) for each editor:	
Editor	Edit Mode
XML Editor	Text
XSD Editor	Design
HTML Editor	Text
WSDL Editor	Text
XSL Editor	Text
NVDL Editor	Text
XProc Editor	Text
RNG Editor	Text
Schematron Editor	Text
<input type="button" value="Edit"/>	

The Oxygen XML Editor plugin “Edit modes” Preferences Page



Text Diagram Preferences

To open the **Diagram** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Text / Diagram**.

- **Show Full Model XML Schema diagram** - If this option is enabled the *old synchronized version* of the schema diagram is available in the Text mode for XML Schemas. For editing in the schema diagram, you can use the *new schema diagram* editor mode.
 - 📄 **Note:** When handling very large schemas, 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** - This option 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** - This option 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 in the editing relative to the diagram **Text** editor.

Grid Preferences

To open the **Grid** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Grid**.

The following preferences are available:

- **Compact representation** - If checked, a more *compact representation* of the grid is used: a child element is displayed at the same height level with the parent element.
 - 📄 **Note:** In the *non-compact representation*, a child element is presented nested with one level in the parent container, one row lower than the parent.
- **Format and indent when passing from grid to text or on save** - The content of the document is formatted by applying the *Format and Indent* action on every switch from the **Grid** editor to the **Text** editor of the same document.
- **Default column width (characters)** - 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** - 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

To open the **Author** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Author**.

The following options are available:

- **Author default background color** - default background color of the Author editing mode. The `background-color` CSS property set in the CSS file associated with the current edited document overwrites this option;
- **Author default foreground color** - default foreground color of the Author editing mode. The `color` CSS property set in the CSS file associated with the current edited document overwrites this option;
- **Show placeholders for empty elements** - when enabled, placeholders are displayed for empty elements to make them clearly visible. A placeholder is rendered as a light grey box displaying the element name;
- **Show Author layout messages** - if enabled, all errors reported during layout creation are presented in the **Errors** view;
- **Show comments** - shows comments of the documents you edit in **Author** mode;
- **Show processing instructions** - shows processing instructions of the documents you edit in **Author** mode;
- **Show doctype** - shows *doctype* sections of the documents you edit in **Author** mode;
- **Display referred 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 refer. 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 of an image beyond which Oxygen XML Editor plugin scales the image;
- **Show very large images** - enables the very large images support (larger than 6 megapixels) in the **Author** mode;
 - ⚠ **Important:** If you enable this option and your document contains many such images, Oxygen XML Editor plugin may consume all available memory, throwing an *OutOfMemory error*. This means that you need to restart the application after you increase the available memory limit.

- **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** - default display mode for element tags presented in **Author** mode. You can choose between:
 - **Full Tags with Attributes** - this mode reveals the entire XML markup, easing the transition from a Text based editing to an **Author** mode editing. Tags contain element names, attribute names, and attribute values;
 - **Full Tags** - displays name tags that enclose block and inline elements;
 - **Block Tags** - displays a mix of element name tags that enclose block elements and compact tags that enclose the inline elements;
 - **Inline Tags** - displays only name tags that enclose the inline elements;
 - **Partial Tags** - displays compact tags that enclose the inline elements;
 - **No Tags** - no tags are displayed. This representation is as close as possible to the document published content;
- **Tags background color** - allows you to configure the Author tags background color;
- **Tags foreground color** - allows you to configure the Author tags foreground color;
- **Whitespaces**
 - **Foreground color** - specifies the foreground color of the whitespaces rendered in the **Author** mode. To enable this option, also select the **Show whitespaces characters** option from **Window > Preferences > Text Editors**.

Caret Navigation Preferences

To open the **Caret Navigation** preferences page, go to **Options > Preferences > Author > Caret Navigation**. The following options are available:

- **Highlight elements near caret** - defines the background color of the element or elements at caret position;
- **Show caret position tooltip** - if enabled, the caret position information tooltip is displayed. More information about the position information tooltip can be found in the section [Position information tooltip](#). The documentation tooltip can be disabled from the [Annotations preferences p](#);
- **Quick up/down navigation** - speeds up navigation when using up / down keys. The cursor jumps from line to line, without stopping at intermediate positions between element tags;
- **Arrow keys move the caret in the writing direction** - controls the caret movement in the bidirectional (BIDI) text for documents opened in the **Author** mode. Pressing the right arrow key increases the offset of the caret in the document. The left one decreases the offset. This means pressing the right arrow key advances the caret in the text in the reading direction. If the option is not selected, pressing the right arrow will simply move the caret to the right, regardless of the text direction.

Schema Aware Preferences

To open the **Schema Aware** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Author > Schema aware**.

- **Schema aware normalization, format, and indent**

When opening a document in Author, white spaces can be normalized or removed in order to obtain a more compact display. The reverse process takes place when saving the document in the Author. By default this algorithm is controlled by the CSS `display` property.

If this option is checked, then this process will be schema aware so the algorithm will take into account if the element is declared as element-only or mixed. It will also take into account options **Preserve space elements**, **Default space elements**, **Mixed content elements** from option page [Window > Preferences > oXygen XML Editor > Editor > Format > XML](#)

- **Indent blocks-only content**

If checked, even if an element is declared in the schema as being mixed but it has a blocks-only content (as specified by the CSS property `display` of its children), it will be treated as being element-only.

- **Schema Aware Editing**

Editing in Author is schema driven.

- **On** - Enables all schema aware editing options.
- **Off** - Disables all schema aware editing options.
- **Custom** -

- **Delete element tags with backspace and delete**

Controls the behavior for deleting element tags using delete or backspace keys. Available options:

- **Smart delete**

If the result of the delete action is invalid, different strategies will be applied in order to keep the document valid. If backspace / delete is pressed at the beginning / end of an element the action that should take place is unwrap (the element will be deleted and its content will be put in its place). If its content is not accepted by the schema in that position, you can keep a valid document by applying different strategies like:

- Search for a preceding (backspace case)/following (delete case) element in which you can append that content.
 - If the tag markers of the element to unwrap are not visible a caret move action in the delete action direction will be performed.

- **Reject action when its result is invalid**

If checked and the result of the delete action is invalid, the action will not be performed.

- **Paste and Drag and Drop**

Controls the behavior for paste and drag and drop actions. Available options:

- **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, different strategies are applied to find an appropriate insert position, like:

- If a sibling element can accept the content, then a new element with the same name as the sibling is created in which the content will be inserted.
 - An analysis will be performed to the left or to the right of the insertion position, without leaving the current context, and try to insert the fragment in one of the encountered elements (that accepts the content to be inserted).
 - If one of the ancestors of the element at caret position accepts the content, after the current offset or before it, then the paste operation will be performed inside this ancestor.

- **Reject action when its result is invalid**

If checked and the result of the paste or drop action is invalid, the action will not be performed.

- **Typing**

Controls the behavior that takes place when typing. Available options:

- **Smart typing**

If the typed character cannot be inserted at element from the caret position then a sibling element that can accept it will be searched for. If the sibling element can accept the content, then a new element with the same name as the sibling is created in which the content will be inserted.

- **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:

- **Allow only insertion of valid elements and attributes**

If checked, only elements or attributes from the content completion proposals list can be inserted in the document through content completion.

- **Show all possible elements in the content completion list**

Specifies whether Oxygen XML Editor plugin displays in the content completion list of proposals all the possible elements. If an element which is not allowed at the current offset is chosen by the user, the application will attempt to insert it in a proper location using the schema aware strategies.

- **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:

- **Delete Element Tags**

If checked, when the *Delete Element Tags* action will result in an invalid content, a warning message will be displayed in which the user is asked if the operation should continue.

- **Join Elements**

If checked, when the *Join Elements* action will result in an invalid content, a warning message will be displayed in which the user is asked if the operation should continue.

- **Convert external content on paste**

If checked, Oxygen XML Editor plugin preserves the formatting style when you paste content copied from external applications (like web browsers or Office-like applications). This option is enabled by default and applies only to the major document type frameworks (DocBook, DITA, TEI, XHTML) and to those customized to support the *content conversion on paste*.

If the Schema Aware Editing is **On** or **Custom** all actions that can generate invalid content will be forwarded first toward *AuthorSchemaAwareEditingHandler*.

Review Preferences

To open the Author **Review** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Author > Review**.

The available preferences are:

- **Author** - The name of the user who performs the changes when **Track Changes** is active for a given editor. This information is associated with each performed change;
- **Track Changes (applies for all authors):**
- **Initial State** - Controls the initial **Track Changes** state;

- **Stored in document** - Recommended when multiple editors work on the same set of documents because the state of **Track Changes** (enabled/disabled) is kept in the edited document. When you open such a document and the **Store in document** option is active, if the `<?oxy_options track_changes="on"?>` processing instruction was saved in the document, the **Track Changes** is enabled. When this option is enabled and you open a document in the **Author** mode, the **Track Changes** state is restored from the previous use of the document. This means that if another user edited the document with **Track Changes** turned on before sending it to you, you will also have **Track Changes** switched on when you open it in the **Author** mode;
- **Always On** - The **Track Changes** feature is active when you open a document;
- **Always Off** - The **Track Changes** feature is inactive when you open a document;



Note: Initial Track Changes State options do not affect documents already open in the **Author** mode.

- **Changed lines marker** - enable this option to display a vertical stripe in the left of the content that is commented, or modified when **Track Changes** is enabled;
- **Inserted content color** - Sets the color used for marking the inserted content. This includes the font foreground and background, and the underline that decorates all inserted text:
 - **Auto** - Automatically assigns colors for the insert changes for each author name. Enabled by default;
 - **Custom** - Uses a custom color for all insert changes, regardless of the author name;
 - **Use same color for text foreground** - Uses the same color to paint the inserted text foreground;
 - **Use same color for background** - Uses the same color for the insert text background. Use the slider to adjust the transparency level;
- **Deleted content color** - Sets the color used for marking the deleted content. This includes the font foreground and background, and the strike-through that decorates all deleted text:
 - **Auto** - Automatically assigns colors for the delete changes for each author name. Enabled by default;
 - **Custom** - Uses a custom color for all delete changes, regardless of the author name;
 - **Use same color for text foreground** - Uses the same color to paint the deleted text foreground;
 - **Use same color for background** - Uses the same color for the deleted text background. Use the slider to adjust the transparency level;
- **Commented color (applies for all authors)** - Sets the background color of the annotated text:
 - **Auto** - Automatically assigns colors for the annotated content for each author name. Enabled by default;
 - **Custom** - Use a custom color for all annotated content, regardless of the author name. Use the slider to control the transparency level.

Callouts Preferences

To open the *Callouts* preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes/Pages > Author > Review > Callouts**. You can also open this preferences page from the contextual menu of a callout. To open the preferences page from the contextual menu, right click the callout and select **Callouts Options**.

This preferences page is divided in two sections. In the first section, you can choose what type of callouts Oxygen XML Editor plugin displays. In the second section, you can customize the parameters of the callouts layout.

The options of the first section are:

- **Comments** - displays author review comment callouts and track changes comment callouts. This option is enabled by default;
- **Track Changes deletion** - enable this option to display deletion callouts;
 - **Show deleted content in callout** - enable this option to display the actual deleted content in the displayed callouts;
- **Track Changes insertion** - enable this option to display insertion callouts.
 - **Show inserted content in callout** - enable this option to display the actual inserted content in the displayed callouts;

The following options of the second section control the amount of information shown in each callout:

- **Show review time** - enable this option to display the date and time of an insertion, deletion, or comment insertion;
- **Show all connecting lines** - enable this option to display the connecting lines between targets and their corresponding callouts;
- **Callouts pane width (px)** - this drop-box specifies the width of a callout. The default value of the **Callouts pane width (px)** field is 200 pixels. Click the drop-box to modify its value.
- **Callouts text limit (characters)** - this drop-box specifies how many characters Oxygen XML Editor plugin displays inside a callout. The default value of the **Callouts text limit (characters)** field is 160. Click the drop-box to modify its value.

Profiling / Conditional Text Preferences

To open the Author **Profiling/Conditional Text** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Author > Profiling/Conditional Text**.

This preferences page contains two sections that present the **profiling attributes** and the **profiling condition sets** defined for the supported document types. You can customize your *profiling attributes* and *condition sets*, or use the ones that Oxygen XML Editor plugin comes predefined with.

In case you have two or more identically named entries that match the same document type, Oxygen XML Editor plugin 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. The **Import from DITAVAL** dialog box contains two tables. The first one displays the attributes and values extracted from the `.ditaval` files and the second one the displays the already defined profiling attributes.



Note: When importing profiling attributes from DITAVAL files, Oxygen XML Editor plugin automatically creates condition sets based on these files.

MathML Preferences

To configure the *MathML* editor of the **Author** mode, go to **Window > Preferences > <oXygen/> Editor > Edit Modes / Pages > Author > MathML**. You can configure the following parameters:

- **Equation minimum font size** - the minimum size of the font used for rendering mathematical symbols;
- **MathFlow installation directory** - the installation folder where Oxygen XML Editor plugin looks for *MathFlow* Components (the *MathFlow* SDK);
- **MathFlow license file** - the license for *MathFlow* Components (the *MathFlow* SDK);
- **MathFlow preferred editor** - a *MathML* formula can be edited in one of three editors of the *MathFlow* Components (*MathFlow* SDK): structure editor, style editor, and simple editor. More documentation is available on the [MathFlow SDK](#) website.

AutoCorrect Preferences

To open the **AutoCorrect** preferences page, go to **Options > Preferences > Editor > Edit Modes > Author > AutoCorrect**.

The following options are available:

- **Replace "double quotes" with "smart quotes"** - enable this option to insert smart quotes instead of double quotes automatically;
- **Replace "single quotes" with "smart quotes"** - enable this option to insert smart quotes instead of single quotes automatically;

To change the symbols that are inserted automatically when these options are enabled, click the **Start quote** and **End quote** buttons and select the symbols you want from the .

Schema Design Preferences

To open the **Schema Design** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Edit modes > Schema Design**

The following options are available in the **Schema Design** preferences page:

- **Show annotation in the diagram** - displays the content of `xs:documentation` elements in the XML Schema **Design** view;
- **When trying to edit components from another schema** - specifies the default behavior when you try to edit a component from an imported schema. You can choose between:
 - **Always go to its definition** - edits the component definition in the imported file;
 - **Never go to its definition** - edits the component definition in the current file;
 - **Always ask** - you are always prompted to choose where you want to edit the component definition;

Properties

You can decide to display additional properties for XML Schema components in the XML Schema **Design** view and customize the properties displayed for each schema component. For each component properties, you can decide if you want to display them only when having a specified value or all the time.

Text Diagram Preferences

To open the **Diagram** preferences page, go to **Options > Preferences > Editor > Edit Modes > Text > Diagram**. The following options are available:

- **Show Full Model XML Schema diagram** - enable this option to make the *old synchronized version* of the schema diagram available in the **Text** mode for XML Schemas. For editing in the schema diagram, you can use the *new schema diagram* editor mode;



Note: When handling very large schemas, 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.

Format Preferences

To open the **Format** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Format**.

The following options are available:

- **Detect indent on open** - the editor detects the indent settings of the open XML document. This way you can correctly format (pretty-print) files that were created with different settings, without changing your options every time you edit such a file. Besides, you can activate the option for detecting the maximum line width used by the formatting and hard wrap mechanism. These features were designed to minimize the differences created by the **Format and Indent** operation when working with a versioning system, like CVS for example;



Note: If the document contains different-size indents, the application computes a weighted average value.

- **Use zero-indent, if detected** - the formatting operation takes into account an indent level of zero in case this is detected.
- **Indent with tabs** - when enabled, sets the indent size to a *tab* unit. When unchecked, the application uses space characters to form an indent. The number of space characters that form a *tab* is defined by the **Indent size** option;
- **Indent size** - sets the number of space characters or the tab size that equals a single indent. An *indent* can be a number of spaces or a tab, selectable using the **Indent With Tabs** option. For example, if set to 4, one tab equals:
 - either 4 space characters, if the **Indent With Tabs** option is unchecked;
 - or one tab that spans 4 characters, if the **Indent With Tabs** option is checked.

- **Hard line wrap** - when enabled, Oxygen XML Editor plugin breaks the edited line automatically when its length exceeds the maximum set line width. This feature allows you to neatly edit a document;
- **Indent on Enter** - if enabled, Oxygen XML Editor plugin indents the new line introduced when pressing the Enter key;
- **Enable Smart Enter** - if you press the Enter key between a start and an end tag, Oxygen XML Editor plugin places the cursor in an indented position on the empty line formed between the start and end tag;
- **Detect line width on open** - detects the line width automatically when the document is opened;
- **Format and indent the document on open** - when enabled, an XML document is formatted and indented before opening it in the editor panel. This option applies only to documents associated with the XML editor.
- **Line width - Format and Indent** - defines the point at which the **Format and Indent** (pretty-print) function 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;
- **Clear undo buffer before Format and Indent** - if checked, you cannot undo anymore editing actions that preceded the **Format and Indent** operation. Only modifications performed after you have performed the operation can be undone. Check this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

To watch our video demonstration about the formatting options offered by Oxygen XML Editor plugin, go to http://oxygenxml.com/demo/Autodetect_Formatting.html.

XML

To open the XML Format preferences page, go to **Window > Preferences > Oxygen XML Editor > 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, **Break long lines** 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, ex. `<a attr1="v1">`. 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 white spaces found in XML content;

- **Preserve space** - List of elements for which the **Format and Indent** operation preserves the whitespaces (like 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 like `xs` are treated as part of the element name without taking into account its binding to a namespace.
- **Default space** - This list contains the names of the elements for which contiguous whitespaces are merged by the **Format and Indent** operation into one blank character.
- **Mixed content** - The elements from this list are treated as mixed 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** - the *Preserve space* elements (identified by the `xml:space` attribute set to `preserve` or by their presence in the **Preserve space** elements list) are normally ignored by the **Format and Indent** operation. When this option is enabled and you are editing 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 indent style of the document you are copying content from, disable this option.

Whitespaces Preferences

This panel displays the special whitespace characters of Unicode. Any character that is checked in this panel is considered whitespace that can be normalized in an XML document. The whitespaces are normalized in any of the following cases:

- when the **Format and Indent** action is applied on an XML document;
- when you switch from **Text** mode to **Author** mode;
- when you switch from **Author** mode to **Text** mode;

The characters with the codes 9 (TAB), 10 (LF), 13 (CR) and 32 (SPACE) are always in the group of whitespace characters that must be normalized so they are always enabled in this panel.

CSS Properties

To open the CSS Format preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Format > CSS**.

The following options control the behavior of the **Format and Indent** operation:

- **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

To open the JavaScript format preferences page, go to **Window > Preferences > oXygen XML Editor > 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 plugin to format embedded JavaScript code, taking precedence over the *Schema aware format and indent* option. This option is enabled by default.

Content Completion Preferences

The *Content Completion Assistant* is a powerful tool that enables inline syntax lookup and auto completion of mark-up elements and attributes. It streamlines mark-up and reduces errors while editing. These settings define the operating mode of the **Content Completion Assistant**.

To open the **Content Completion** preferences page, go to **Window > Preferences > Oxygen XML Editor > Editor > Content Completion**.

The following options are available:

- **Auto close the last opened tag** - Oxygen XML Editor plugin closes the last open tag when you type `</`;
- **Automatically rename/delete/comment matching tag** - Oxygen XML Editor plugin automatically mirrors the matching end tag when you rename a start tag. You are also able to delete one or more start tags and their matching tags are deleted automatically. This is available for the **Toggle comment** option as well.



Note: In case you select **Toggle comment** for multiple starting tags (or you delete them) and the matching end tags area interferes with start tags, the end tags are not commented (or deleted).

- **Use content completion** - activates the content completion assistant;
- **Close the inserted element** - when you choose an entry from the **Content Completion Assistant** list of proposals, Oxygen XML Editor plugin 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 plugin inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is *associated with the edited XML document*. This option is applied also in the **Author** mode of the XML editor;
 - **Add optional content** - Oxygen XML Editor plugin inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema. This option is applied also in the **Author** mode of the XML editor;
 - **Add first Choice particle** - Oxygen XML Editor plugin inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema. This option is applied also in the **Author** mode of the XML editor;
- **Case sensitive search** - the search in the content completion assistant window when you type a character is case-sensitive ('a' and 'A' are different characters). This option is applied also in the **Author** mode of the XML editor;
- **Cursor position between tags** - when enabled, Oxygen XML Editor plugin sets the cursor automatically between start and end tag for:
 - 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 plugin 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 plugin 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 plugin 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 plugin 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 their icon is decorated with a small red square.. 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 plugin 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 plugin 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 (Meta on Mac OS) + Space**) - When selected, Oxygen XML Editor plugin learns the typed words and makes them available in a content completion fashion by pressing **Ctrl (Meta on Mac OS) + Space** on your keyboard;



Note: In order to be learned, the words need to be separated by space characters.

Annotations Preferences

To open the **Annotations** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Content Completion > Annotations**.

The following preferences can be configured for the annotations of the elements and attributes displayed by the **Content Completion Assistant**:

- **Show annotations** - Oxygen XML Editor plugin displays the schema annotations of an element, attribute, or attribute value currently selected in the **Content Completion Assistant** proposals list. This option is applied also in the **Author** mode of the XML editor.
- **Show annotations as tooltip** - Oxygen XML Editor plugin displays the annotation of elements and attributes as a tooltip when you hover over them with the cursor in the XML editor panel or in the **Elements** view (both *the Text editing mode one* and *the Author editing mode one*). This option is applied also in the **Author** mode of the XML editor.
- **Use DTD comments as annotation** - When enabled, Oxygen XML Editor plugin uses all DTD comments as annotations. If this option is disabled, Oxygen XML Editor plugin displays only special Oxygen XML Editor plugin `doc:` comments, or if they are missing, it displays any other comment found in the DTD.
- **Use all Relax NG annotations as documentation** - When enabled, 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 disabled, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

XSL Preferences

The **XSL** preferences page defines the elements that the **Content Completion Assistant** suggests for the XSL editor in addition to the XSL elements. To open the **XSL** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Content Completion > XSL**.

The following options are available:

- **Automatically detect XHTML transitional or Formatting objects** - detects if the XSL uses the XHTML or FO schemas to drive the **Content Completion Assistant**. Oxygen XML Editor plugin analyzes the namespaces declared in the root element to find an appropriate schema.

If the detection fails, Oxygen XML Editor plugin uses one of the following options, depending on your choice:

- **None** - the **Content Completion Assistant** offers only the XSL 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 (FO) elements as substitutes for `xsl:element`;
- **Custom schema** - the **Content Completion Assistant** includes elements from a 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

To configure the options for the content completion in XPath expressions, go to **Window > Preferences > oXygen XML Editor > Editor > Content Completion > XPath**.

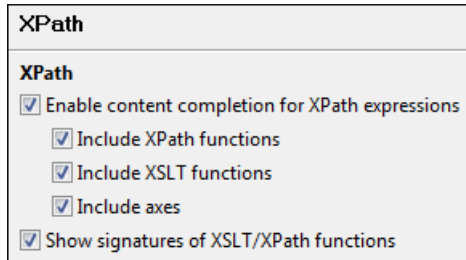


Figure 336: The Content Completion XPath Preferences Panel

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. Options are available to control if the XPath functions, XSLT functions and XSLT axes are presented in the content completion list of proposals when editing XPath expressions.
- **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.

XSD Preferences

To open the XSD preferences page, go to **Window > Preferences > oXygen XML Editor > 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

-  **Note:**

Syntax Highlight Preferences

Oxygen XML Editor plugin supports syntax highlight for XML, DTD, Relax NG (XML and Compact Syntax), Java, JavaScript / JSON, PHP, CSS, XQuery, C++, C, Perl, Properties, SQL, Shell and Batch documents. While Oxygen XML Editor plugin provides a default color configuration for highlighting the tokens, you can choose to customize it, using the Syntax Highlight options panel.

To open the Syntax Highlight preferences page go to **Window > Preferences > oXygen XML Editor > Editor > Syntax Highlight**.

Each document type has an associated set of tokens. When a document type node is expanded, the associated tokens are listed. For each token, you can customize the color and the font style. These properties are used in **Text** mode of the editor panel. The tokens for XML documents are used also in XSD, XSL, RNG documents.



Note: The **Preview** area contains 4 tabs that allow you to preview XML, XSD, XSL, RNG sample files as they are rendered in Oxygen XML Editor plugin.

When you do not know the name of the token that you want to configure, select a token by clicking directly on that type of token in the **Preview** area.

You can edit the following color properties of the selected token:

- **Foreground** - The **Foreground** button opens a color dialog that allows setting the color properties for the selected token with one of the following color models: Swatches, HSB, or RGB;



Note: You can also open the dialog for changing the foreground color of a token by double-clicking (or pressing *Enter*) on the tree node that corresponds to that token.

- **Background** - The **Background** button opens the same color dialog as the **Foreground** button;
- **'<' & '>' color** - available only for **XML > Tag**, this option specifies the color of the < and > XML tags;
- **Bold style** - This checkbox enables the bold variant of the font for the selected token. This property is not applied to a bidirectional document;
- **Italic style** - This checkbox enables the italic variant of the font for the selected token. This property is not applied to a bidirectional document.

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

The **Elements / Attributes by Prefix** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > Editor > Syntax Highlight > Elements / Attributes by Prefix**.

One row of the table contains the association between a namespace prefix and the properties to mark start tags and end tags, or attribute names in that prefix. Note that the marking mechanism does not look at the namespace bound to that prefix. If the prefix is bound to different namespaces in different XML elements of the same file, all the tags and attribute names with that prefix are marked with the same color.

You can edit the following color properties of the selected token:

- **Foreground** - The **Foreground** button opens a color dialog that allows setting the color properties for the selected token with one of the following color models: Swatches, HSB, or RGB;



Note: You can also open the dialog for changing the foreground color of a token by double-clicking (or pressing *Enter*) on the tree node that corresponds to that token.

- **Background** - The **Background** button opens the same color dialog as the **Foreground** button;

You can choose that only the prefix is displayed with the selected color by enabling the **Draw only the prefix with a separate color** option.

Open / Save Preferences

To open the **Open / Save** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Open / Save**.

The following options are available:

- **Safe save (only for local files)** - option that provides an increased degree of protection in the unlikely event of a failure of the **Save** action. This mechanism creates a temporary file that holds the edited content until it is safely saved in the original file. If the **Save** action fails, the temporary file is kept in the system temporary folder, **OxygenXMLTemp** subfolder.
- **Save all files before transformation or validation** - Saves all open files before validating or transforming an XML document. This way the dependencies are resolved, for example when modifying both the XML document and its XML Schema.

- **Clear undo buffer on save** - If enabled, Oxygen XML Editor plugin erases its undo stack when you save a document. Only modifications made after you have saved the document can be undone. Check this option if you encounter frequent *out of memory* problems (**OutOfMemoryError**) when editing very large documents.

Templates Preferences

This panel groups the preferences that are related with code templates and document templates:

- [Code Templates](#)
- [Document Templates](#)

Code Templates Preferences

Code templates are small document fragments that can be inserted quickly at the editing position and can be reused in other editing sessions. Oxygen XML Editor plugin comes with a large set of ready-to-use templates for XSL, XQuery, and XML Schema. You can even share your code templates with your colleagues using the template export and import functions.

To obtain the template list, use:

- The shortcut key that activates the **Content Completion Assistant** on request is **Ctrl (Meta on Mac OS) + Space**. It displays the code templates names in *the content completion assistant list* together with proposed element names.
- The shortcut key that activates code templates assistant on request: **Ctrl (Meta on Mac OS) + Shift + Space**. It displays only the code templates in the proposals list.

To open the **Code Templates** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Templates > Code Templates**. This preferences page contains a list with all available code templates (both built-in and custom created ones) and a code preview area. You can disable any code template by unchecking its corresponding option box.

- **New** - defines a new code template. You can choose to set the newly defined code template for a specific type of editor or for all editor types;
- **Edit** - edits the selected code template;
- **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;
- **Import** - imports a file with code templates that was created by the **Export** action;
- **Export** - exports a file with code templates.

Document Templates Preferences

The list of document templates that are displayed in *the New dialog* can be extended with custom templates that are specified in the **Document Templates** preferences panel. Add the template files in a folder that is specified in this panel or in the `templates` folder of the Oxygen XML Editor plugin install directory.

To open the **Document Templates** preferences page, go to **Window > Preferences > oXygen XML Editor > 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 plugin looks into these location folders by using the **Up** and **Down** buttons on a selected table row.

Spell Check Preferences

To open the **Spell Check** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Spell Check**.

The following options are available:

- **Spell checking engine** - the application ships with two spell check engines, *Hunspell* and *Java spell checker*. Each engine has a specific format of spelling dictionaries. The languages of the built-in dictionaries of the selected engine are listed in the **Default language** options list.



Note: In case you configured an additional location for the dictionaries that Oxygen XML Editor plugin uses, the **Default language** option collects the languages both from the default and additional locations.

- **Automatic Spell Check** - enable this option to make the spellchecker highlight errors as you edit a document;
- **Select editors** - allows you to select the file types for which the automatic spell check takes effect. File types in which automatic spell check is generally not useful, like CSS and DTD, are excluded by default.
- **Default language** - the default language list allows you to choose the language used by the spell check engine;



Note: You can *add more spelling dictionaries* to the spell check engines.

- **Use "lang" and "xml:lang" attributes** - if enabled, the contents of any element with one of the `lang` or `xml:lang` attributes is checked using a dictionary suitable for the language specified in the attribute value, if this dictionary is available. When these attributes are missing, the language used is controlled by the two radio buttons: **Use the default language** or **Do not check**;
- **XML spell checking in** - these options allow you to specify if the spell checker will be enabled inside XML comments, attribute values, text, and CDATA sections;
- **Case sensitive** - when enabled, spell checking reports capitalization errors, for example a word that starts with lowercase after *etc.* or *i.e.*;
- **Ignore mixed case words** - when enabled, operations do not check words containing mixed case characters (e.g. *SpellChecker*);
- **Ignore words with digits** - when enabled, the spell checker does not check words containing digits (e.g. *b2b*);
- **Ignore Duplicates** - when enabled, the spell checker does not signal two successive identical words as an error;
- **Ignore URL** - when enabled, ignores words looking like URL or file names (e.g. *www.oxygenxml.com* or *c:\boot.ini*);
- **Check punctuation** - when enabled, punctuation checking is enabled: misplaced white space and wrong sequences, like a dot following a comma, are highlighted as errors;
- **Allow compounds words** - when enabled, all words formed by concatenating two legal words with a hyphen (hyphenated compounds) are accepted. If the language allows it, two words concatenated without hyphen (closed compounds) are also accepted.
- **Allow general prefixes** - when enabled, a word formed by concatenating a registered 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 enabled, accepts any word ending with registered file extensions (e.g. *myfile.txt*, *index.html*, etc.).
- **Ignore acronyms** - when enabled, the acronyms are not reported as errors.
- **Ignore elements** - a list of XPath expressions for the elements that the spell check engine ignores. The following restricted set of XPath expressions are supported:
 - '/' and '/' separators;
 - '*' wildcard.

An example of allowed XPath expression: `/a*/b`.

To change the color used by the spell check engine to highlight spelling errors, go to **Window > Preferences > General > Editors > Text Editors > Annotations > Spell Check Annotation**.

Dictionaries Preferences




To open the **Dictionaries** preferences page, go to **Options > Preferences > Editor > Spell Check > Dictionaries**. This page allows you to configure the dictionaries and term lists that Oxygen XML Editor plugin uses and to choose where to save new learned words.



Note: A term list must have the `.tdi` extension.

The following options are available:

- **Dictionaries and term lists default folder** - specifies the default location where the dictionaries and term lists that Oxygen XML Editor plugin uses are stored;

- **Include dictionaries and term list from** - specifies a location where you can store dictionaries and term lists, different from the default one;
 -  **Note:** In case the additional location contains a file with the same name as one from the default location, the additional file is proffered.
 -  **Note:** The spell checker takes into account term lists, collected both from the default and additional locations.
- **Save learned words in the following location** - specifies the target in which new learned words are saved;
 - **Default** - the default location where the dictionaries and term lists are stored;
 - **Custom** - a custom location where you can store dictionaries and term lists;
- **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.
 -  **Note:** These options are valid when Oxygen XML Editor plugin uses the Hunspell spell checking engine.

Document Checking Preferences

To open the **Document Checking** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Document Checking**. This preferences page contains preferences for configuring how a document is checked for both well-formedness errors and validation errors.

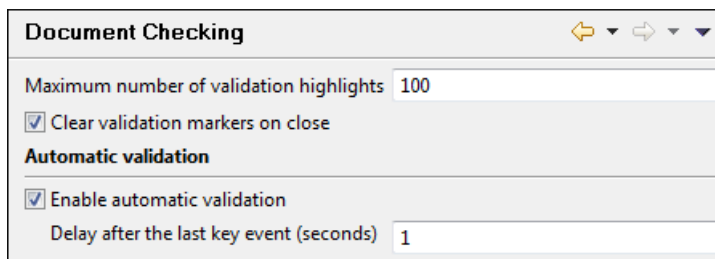


Figure 337: Document Checking Preferences Panel

The error checking preferences are the following:

- **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*.
- **Clear validation markers on close** - If this option is selected all the error markers added in the **Problems** view for that document are removed when a document edited with the Oxygen XML Editor plugin is closed.
- **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen XML Editor plugin.
- **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

Mark Occurrences Preferences

To open the **Mark Occurrences** preferences page, go to **Window > Preferences > oXygen XML Editor > Editor > Mark Occurrences**:

The following preferences are available in this preferences page:

- **XML files** - activates the *Highlight IDs Occurrences* in XML files;
- **XSLT files** - activates the *Highlight Component Occurrences* in XSLT files;
- **XML Schema files and WSDL files** - activates the *Highlight Component Occurrences* in XSD and WSDL files;
- **Declaration highlight color** - color used to highlight the component declaration;
- **Reference highlight color** - color used to highlight component references.

Custom Validation Engines Preferences

The **Custom Validation Engines** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > Editor > Custom Validations**.

If you want to add a new custom validation tool or edit the properties of an existing one you can use the **Custom Validator** dialog displayed by pressing the **New** button or the **Edit** button.

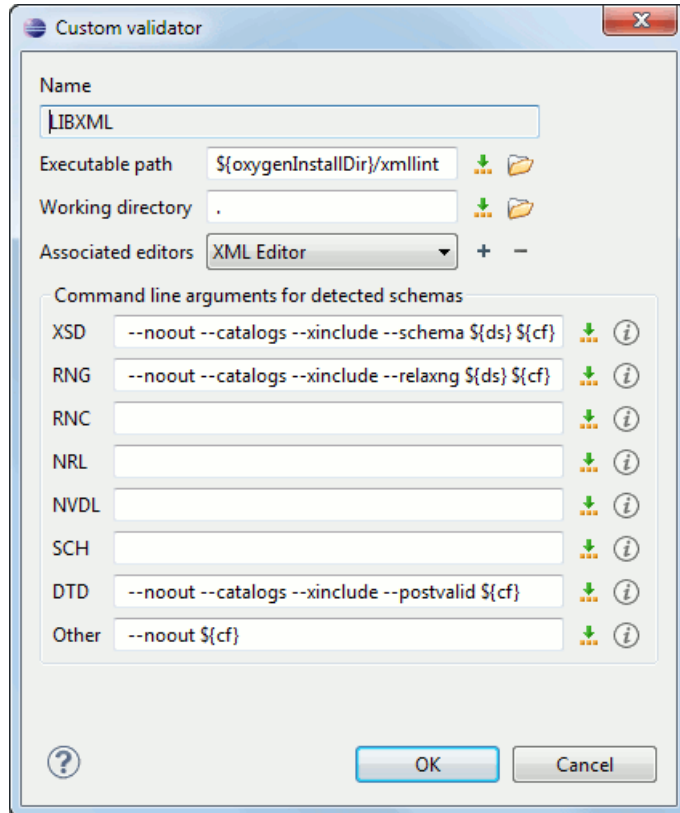


Figure 338: Edit a Custom Validator

The configurable parameters of a custom validator are the following:

- **Name** - Name of the custom validation tool displayed in the **Custom Validation Engines** toolbar.
- **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;
 - `${cfu}` - The path of the current file as a URL. The current file is the one currently opened and selected;
 - `${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 $1 \times 1024 \times 1024 = 1048576$;
- 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 open the **CSS Validator** preferences page, go to **Window > Preferences > oXygen XML Editor > CSS Validator**.

You can configure the following options for the built-in CSS validator of Oxygen XML Editor plugin:

- **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 and 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** - Here 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 plugin 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 open the **XML Catalog** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XML Catalog**.

The following options are available:

- **Prefer** is used for specifying if Oxygen XML Editor plugin tries to resolve first the **PUBLIC** or **SYSTEM** reference from the **DOCTYPE** declaration of the XML document. If **PUBLIC** is preferred and a **PUBLIC** reference is not mapped in any of the XML catalogs then a **SYSTEM** reference is looked up.
- 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 plugin 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 plugin 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 plugin 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 open the **XML Parser** preferences page, go to **Window > Preferences > oXygen XML Editor > 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 referred 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 referring 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 plugin 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 open the **XML Schema** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XML Parser > XML Schema**.

This preferences page allows you to configure the following options:

Xerces validation features

- **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 to Saxon EE;
- **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 XSD 1.1 CTA full XPath 2.0 checking (<http://apache.org/xml/features/validation/cta-full-xpath-checking>)** - controls whether full XPath or an XPath subset is used to evaluate CTA instructions for XSD 1.1;
- **Enable comments and PIs in XDM model, for XSD 1.1 assertion processing (<http://apache.org/xml/features/validation/assert-comments-and-pi-checking>)** - controls whether comments and processing instructions are visible to the XPath expression used to define an assertion for 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 open the **Relax NG** preferences page, go to **Window > Preferences > oXygen XML Editor > 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 open the **Schematron** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XML Parser > Relax NG**.

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.

XML Instances Generator Preferences

The **XML Instances Generator** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > 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 plugin 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.

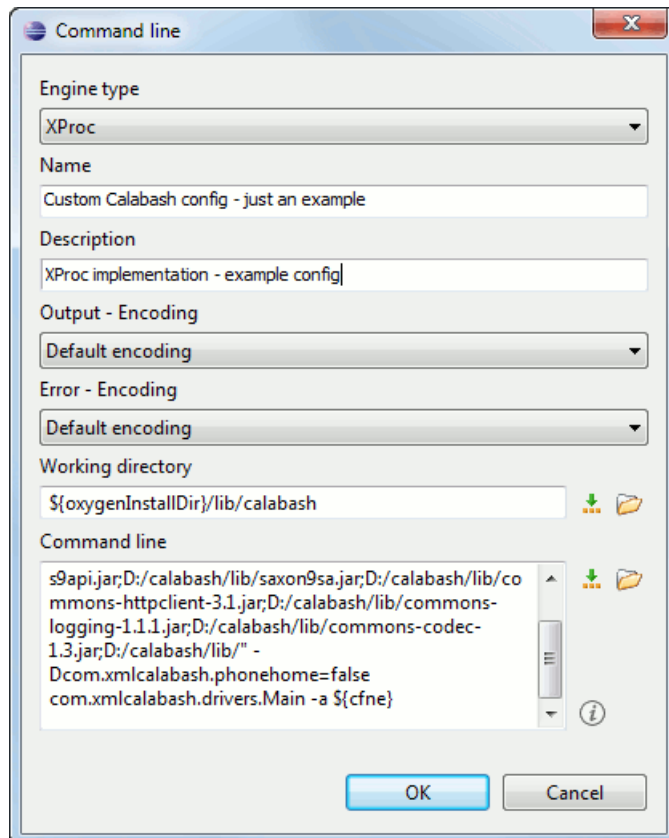


Figure 339: Creating an XProc external engine

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 parameterizing 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/documentation/configuration/configuration-file.xml>.



Note: You can configure Calabash using the `calabash.config` file.



Note: These files are located in `[Oxygen installation directory]\lib\xproc\calabash`. In case they do not exist, you have to create them.

XSLT-FO-XQuery Preferences

The **XSLT/FO/XQuery** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > 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 open the **XSLT** preferences panel, go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XSLT**.

Oxygen XML Editor plugin gives you the possibility to use an XSLT transformer implemented in Java (other than the XSLT transformers that come bundled with Oxygen XML Editor plugin). To use a different XSLT transformer, specify the name of the transformer factory class. Oxygen XML Editor plugin 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 open the **Saxon 6** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon 6**.

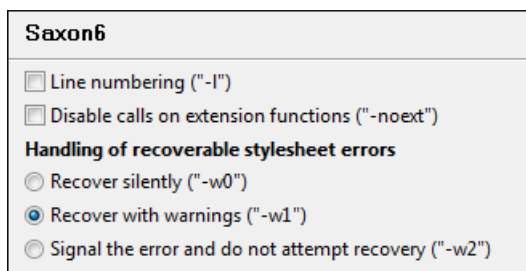


Figure 340: The Saxon 6 XSLT Preferences Panel

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 open the Saxon HE/PE/EE preferences panel go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE**.

Oxygen XML Editor plugin allows you to configure the following XSLT options for the Saxon 9.5.0.1 transformer (editions: Home Edition, Professional, and Enterprise):

- **Use a configuration file ("-config")** - sets a Saxon 9 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 number is included in the output messages;
- **Debugger trace into XPath expressions (applies to debugging sessions)** - instructs the *XSLT Debugger* to step into XPath expressions;
- **Expand attributes defaults ("-expand")** - specifies whether the attributes defined in the associated DTD or XML Schema that have default values are expanded in output of the transformation you are executing;
- **DTD validation of the source ("-dtd")** - the following options are available:
 - **On** - requests *DTD-based* 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, because 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")**;
 - **Recover with warnings ("recover")** - default setting;
 - **Signal the error and do not attempt recovery ("fatal")**.
- **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, whitespaces are stripped if this is specified in the stylesheet using `xsl:strip-space`.
- **Optimization level ("-opt")** - set optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization); currently all values other than 0 result in full optimization but this is likely to change in the future.

The default is full optimization; this feature allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably. (Note however, that even with no optimization, the lazy evaluation may still cause the evaluation order to be not as expected).

The advanced options available only in Saxon PE / 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, perhaps 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;

The advanced options available only in Saxon EE are:

- **XML Schema version** - use this option to change the default XML Schema version set in **Options > Preferences > 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. Validation is available only with Saxon-EE, and this flag automatically switches on the `-sa` option. Available options:
 - **Schema validation ("strict")** - this mode requires an XML Schema and specifies that the source documents should be parsed with schema-validation enabled;
 - **Lax schema validation ("lax")** - this mode specifies if the source documents should be parsed with schema-validation enabled if an XML Schema is provided;
 - **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)")** - when 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 treatment. For further details regarding this option, go to <http://www.saxonica.com/documentation/javadoc/>;
- **Initializer class** - equivalent with 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.

Saxon HE/PE/EE Advanced Preferences

To open the **Saxon HE/PE/EE Advanced** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE > Advanced**.

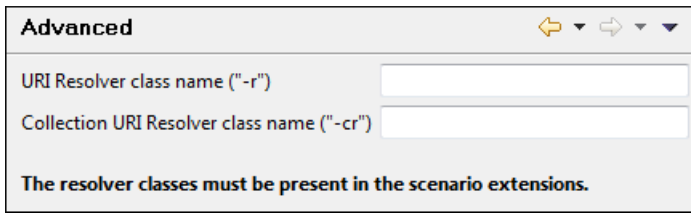


Figure 341: The Saxon HE/PE/EE XSLT Advanced Preferences Panel

You can configure the following advanced XSLT options for the Saxon 9.5.0.1 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 9.5.0.1 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 9.5.0.1 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

The XSLTProc preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > 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 plugin 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 plugin will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen XML Editor plugin 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 plugin 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

The MSXML preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > 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

The MSXML.NET preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > 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.XslTransform` 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 open the **XQuery** preferences page, go to **Window > Preferences > Oxygen XML Editor > 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 plugin 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 exceeded, 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 plugin 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 open the **Saxon HE/PE/EE** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE**.

The XQuery preferences for the Saxon 9.5.0.1 are the following:

- **Use a configuration file ("-config")** - sets a Saxon 9 configuration file that is used for XQuery transformation and validation;
- **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.
- **Strip whitespaces** - can have one of the following three values:
 - **All** - strips all whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document;
 - **Ignore** - 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** - strips no whitespace before further processing.
- **Optimization level** - this option allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably.
 - **Use linked tree model** - this option activates the linked tree model. Enable this option if you are using XQuery Update.

The Saxon 9.5.0.1 Professional Edition options are the following:

- **Disable calls on extension functions** - if unchecked, external functions calls is allowed. 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.

The Saxon 9.5.0.1 Enterprise Edition specific options are the following:

- **Validation of the source** - this determines whether XML source documents should be parsed with schema-validation enabled;
- **Validation errors in the results tree treated as warnings** - available only for Saxon EE. If checked, all validation errors are treated as warnings, otherwise they are treated as fatal;
- **Generate bytecode ("--generateByteCode:(on|off)")** - when 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 treatment. For further details regarding this option, go to <http://www.saxonica.com/documentation/javadoc/>.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - if checked, Saxon EE runs the XQuery transformation with the XQuery 3.0 support;



Note: In case the XQuery 3.0 support is active, the XQuery Update support is no longer available.

- **Backup files updated by XQuery ("-backup:(on|off)")** - if checked, backup versions for any XML files updated with XQuery Update are generated.

Saxon HE/PE/EE Advanced Preferences

The **Saxon HE/PE/EE Advanced** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE > Advanced**.

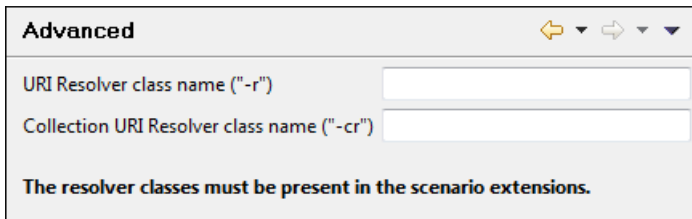


Figure 342: The Saxon HE/PE/EE XQuery Advanced Preferences Panel

The advanced XQuery options which can be configured for the Saxon 9.5.0.1 XQuery transformer (all editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **URI Resolver class name** - Allows the user to specify a custom implementation for the URI resolver used by the XQuery Saxon 9.5.0.1 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.
- **Collection URI Resolver class name** - Allows the user to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.5.0.1 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 open the **Debugger** preferences page, go to **Window > Preferences > oXygen XML Editor > 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;
- **XWatch evaluation timeout (seconds)** - specifies the maximum time that Oxygen XML Editor plugin 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.

Annotations Preferences

To open the **Annotations** preferences page go to **Window > Preferences > General > Editors > Text Editors > Annotations**.

The following Oxygen XML Editor plugin preferences are available:

- **XSLT/XQuery Debug Current Instruction Pointer** - Controls the background color of the current execution node, both in the document (XML) and XSL/XQuery views.

Profiler Preferences

This section explains the settings available for the XSLT Profiler. To access and modify these settings, go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > Profiler** (see [Debugger Preferences](#) on page 737).

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 plugin 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 plugin 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 open the **FO Processors** preferences panel, go to **Window > Preferences > Oxygen XML Editor > XML > XSLT/FO/XQuery > FO Processors**.

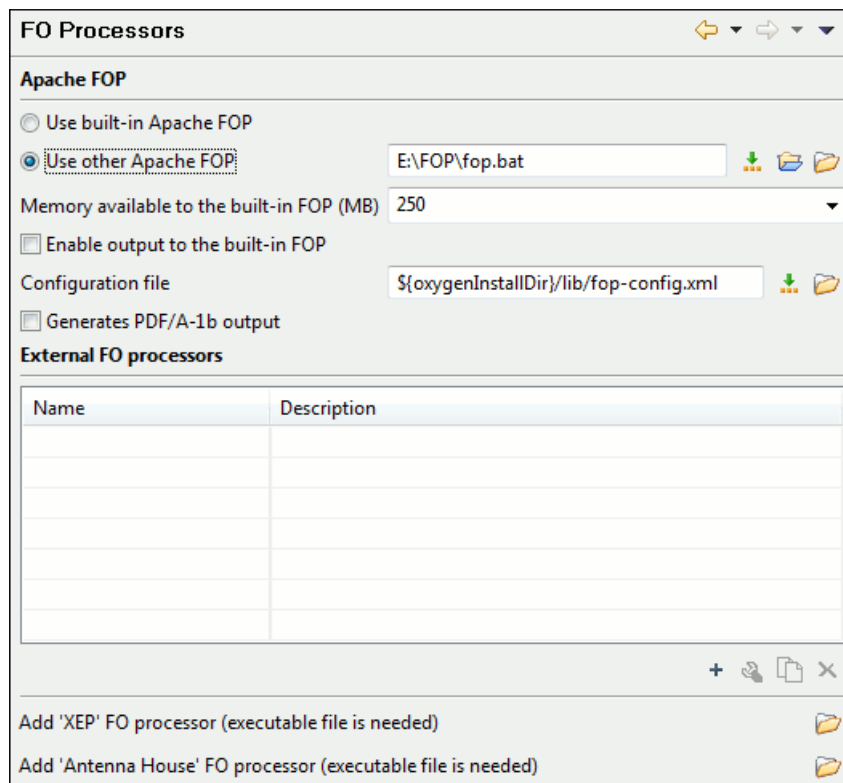


Figure 343: The FO Processors Preferences Panel

Apache FOP

The options for FO processors are the following:

- **Use built-in Apache FOP** - instructs Oxygen XML Editor plugin to use its built-in Apache FO processor;
- **Use other Apache FOP** - instructs Oxygen XML Editor plugin 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 plugin 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:

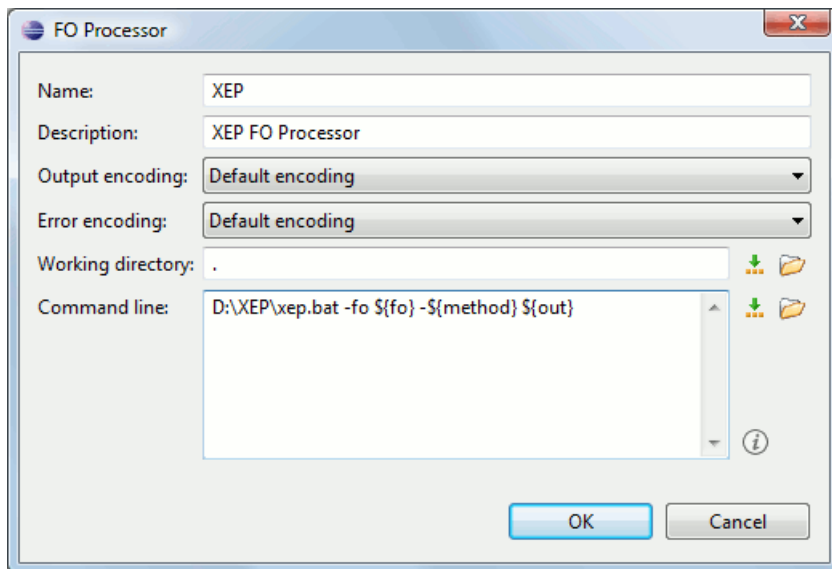


Figure 344: The External FO Processor Configuration Dialog

- **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;
- **Output Encoding** - the encoding of the FO processor output stream displayed in a results panel at the bottom of the Oxygen XML Editor plugin window;

- **Error Encoding** - the encoding of the FO processor error stream displayed in a results panel at the bottom of the Oxygen XML Editor plugin window;
- **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;
 - **`\${cfd}** - 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 plugin 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 plugin install directory;
 - **`\${oxygenInstallDir}** - the Oxygen XML Editor plugin installation directory;
 - **`\${ps}** - the platform-specific path separator. It is used between the library files specified in the class path of the command line.

XPath Preferences

To open the **XPath** preferences panel go to **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XPath**.

Oxygen XML Editor plugin allows you to customize the following options:

- **Unescape XPath expression** - the entities of an XPath expressions that you type in *the XPath dialog* are unescaped. For example the expression


```
//varlistentry[starts-with(@os,'&#x73;')]
```

 is equivalent with:


```
//varlistentry[starts-with(@os,'s')]
```
- **No namespace** - if you enable this option, Oxygen XML Editor plugin considers unprefixed element names of the XPath 2.0 / 3.0 expressions evaluated in *the XPath dialog* as belonging to no namespace.
- **Use the default namespace from the root element** - if you enable this option, Oxygen XML Editor plugin considers unprefixed element names of the XPath expressions evaluated in *the XPath dialog* 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 plugin considers unprefixed element names of the XPath expressions evaluated in *the XPath dialog* 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 used in *the XPath dialog*.
- **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 plugin distribution*.



Note: You can not use these custom engines in *the Debugger perspective*.

To open the **Custom Engines** preferences page, go to **Window > Preferences > oXygen XML Editor > XML > XSLT/FO/XQuery > Custom Engines**.

The following parameters can be configured for a custom engine:

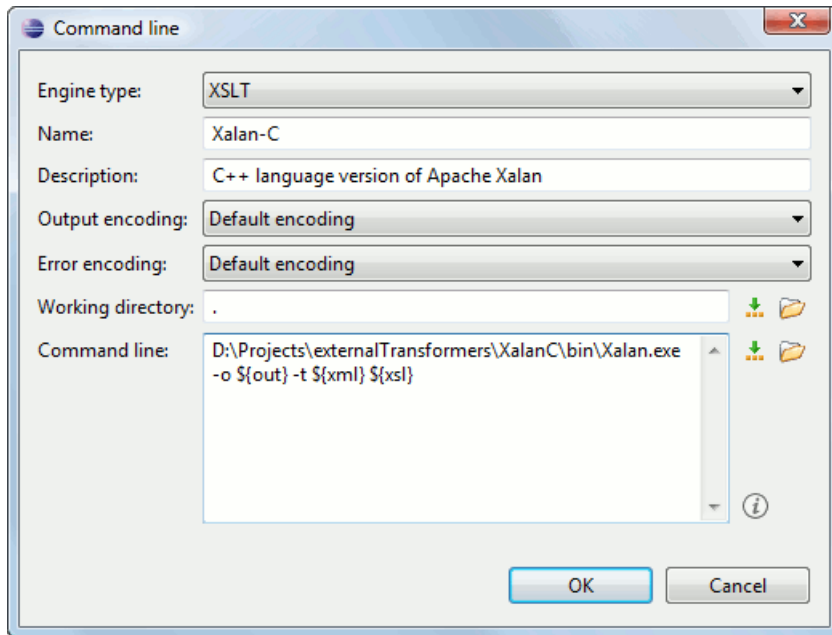


Figure 345: Parameters of a Custom Engine

- **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.
- **Output Encoding** - The encoding of the transformer output stream.
- **Error Encoding** - The encoding of the transformer error stream.
- **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.
 - **\${cfd}** - The path to the directory of the current file.
 - **\${pd}** - The path to the directory of the current project.
 - **\${oxygenInstallDir}** - The Oxygen XML Editor plugin install directory.
- **Command line** - The command line that must be executed by Oxygen XML Editor plugin 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.

Import Preferences

To open the **Import** preferences page, go to **Window > Preferences > oXygen XML Editor > 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 plugin 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 16: 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
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

Letter	Date or Time Component	Presentation	Examples
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.

Data Sources Preferences

To open the **Data Sources** preferences page, go to **Window > Preferences > oXygen XML Editor > Data Sources**.

Data Sources Preferences

To open the **Data Sources** preferences page, go to **Options > Preferences > 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) available for the major database servers.

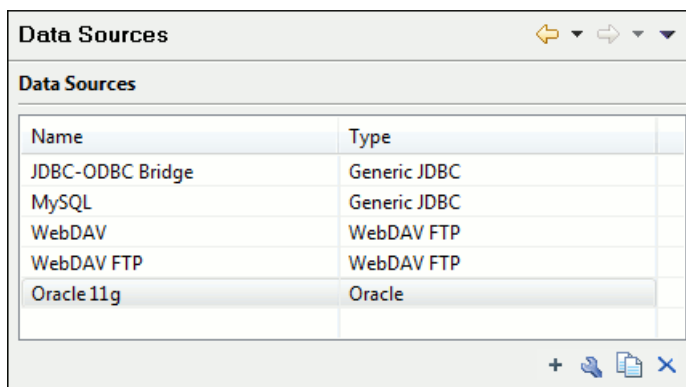


Figure 346: The Data Sources Preferences Panel

- **New** - opens the **Data Sources Drivers** dialog that allows you to configure a new database driver.

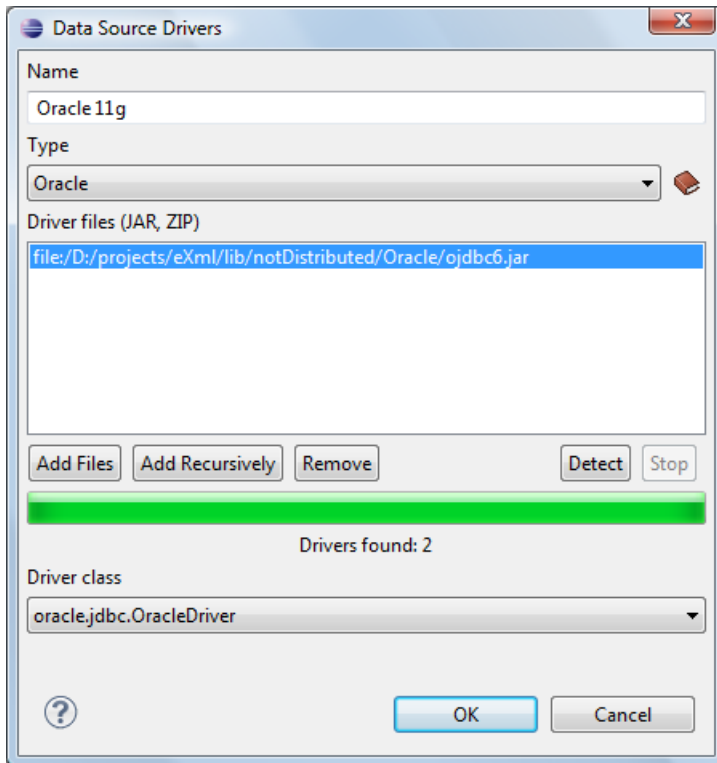


Figure 347: The Data Sources Drivers Dialog

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;

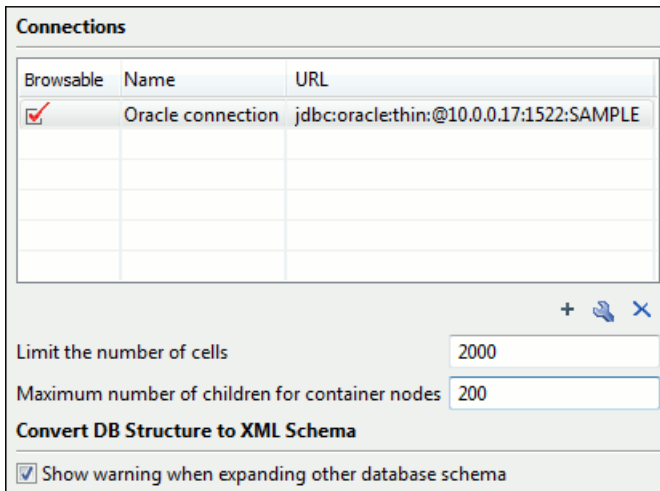


Figure 348: The Connections Preferences Panel

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 **Show warning when expanding other database schema** option controls if a warning message will be displayed when expanding another database schema and there are tables selected in the current expanded one. This applies for the dialog **Select database table** when invoking the **Convert DB Structure to XML Schema** action.

The actions of the buttons from the **Connections** panel are the following:

- **New** - opens the **Connection** dialog which has the following fields:

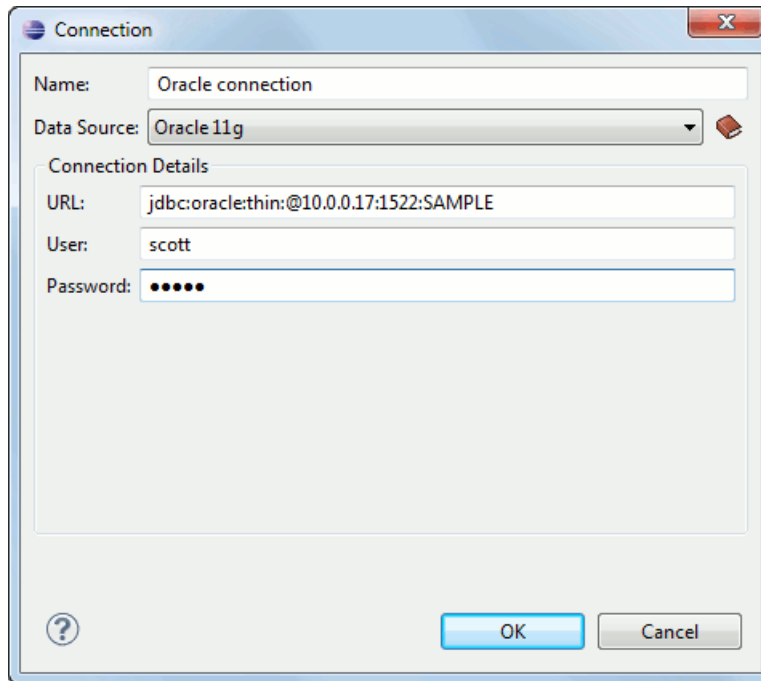


Figure 349: The Connection Dialog

- **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

You can find below the locations where you have to go to get the drivers necessary for accessing databases in Oxygen XML Editor plugin.

- **Berkeley DB XML database** - Copy the jar files from the Berkeley database install directory to the Oxygen XML Editor plugin 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 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 plugin for [configuring a DB2 data source](#).

- **eXist database** - Copy the jar files from the eXist database install directory to the Oxygen XML Editor plugin 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** - Both SQL Server 2005 and SQL Server 2008 are supported. For connecting to SQL Server 2005 you have to download the SQL Server 2005 JDBC driver file `sqljdbc.jar` from the [Microsoft website](#) and use it for [configuring an SQL Server data source](#). For connecting to SQL Server 2008 you have to download the SQL Server 2008 JDBC 1.2 driver file `sqljdbc_1.2\enu\sqljdbc.jar` from the [Microsoft website](#) and use it for [configuring an SQL Server data source](#). Please note that the SQL Server driver is compiled with a Java 1.6 compiler so you need to run Oxygen XML Editor plugin with a Java 1.6 virtual machine in order to use this driver.
- **Oracle 11g database** - Download the Oracle 11g JDBC driver called `ojdbc5.jar` from the [Oracle website](#) and use it for [configuring an Oracle data source](#).
- **PostgreSQL 8.3 database** - Download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar` from the [PostgreSQL website](#) and use it for [configuring a PostgreSQL data source](#).
- **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 plugin install directory as described in [the procedure](#) for configuring a Documentum xDb (X-Hive/DB) 10 data source.

Table Filters Preferences

The **Table Filters** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > Data Sources > Table Filters**.

Here you can choose which of the database table types will be displayed in the **Data Source Explorer** view.

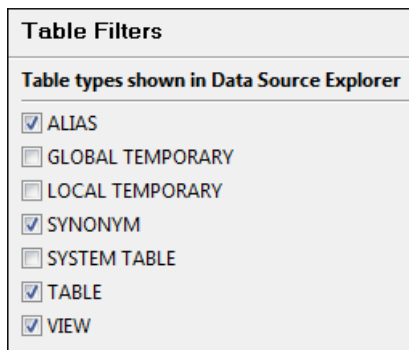


Figure 350: Table Filters Preferences Panel

Archive Preferences

The **Archive** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > 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.

- **Show archive backup dialog** - Select this option if you want to be notified for backup when modifying in archives. The last backup option you chose will always be used as the default one.
- **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 plugin. You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.

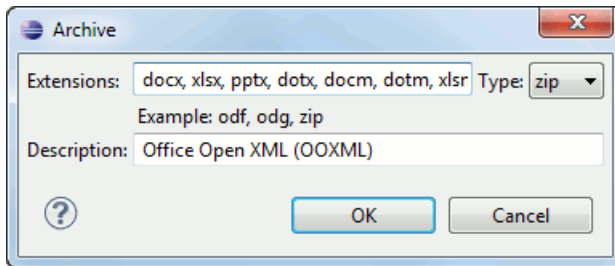


Figure 351: Edit Archive Extension Mappings

Important: You have to restart Oxygen XML Editor plugin 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 an archive is modified in any way with this option turned on, UTF-8 characters are used in the names of all files in the archive.

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.

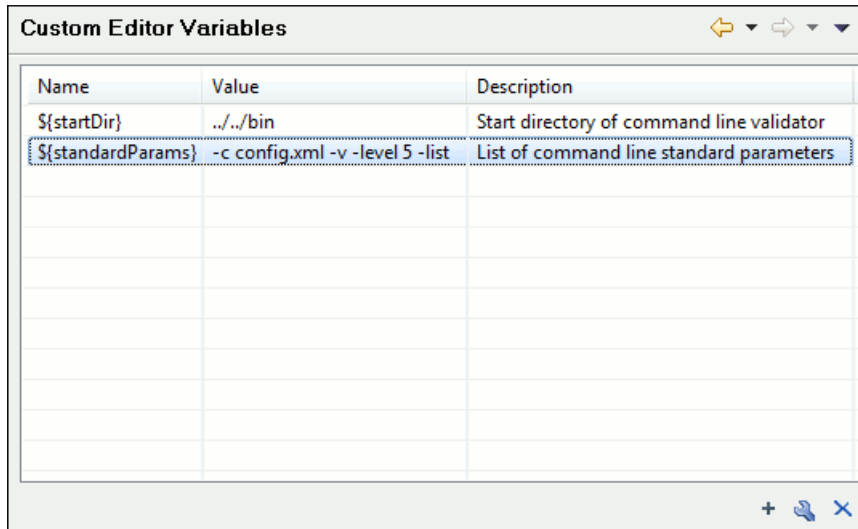


Figure 352: Custom Editor Variables

The Network Connection Settings Preferences

This section presents the options available in the **Network Connection Settings** preferences pages.

HTTP(S) Preferences

To open the **HTTP(S)/WebDAV** preferences page, go to **Options > Preferences > Connection settings > HTTP(S)/WebDAV**. The following options are available:

- **Enable the HTTP(S)/WebDAV Protocols** - .
- **Internal Apache HttpClient Version** - Oxygen uses the Apache HttpClient to establish connections to HTTP servers. To enable oXygen 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, 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 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, 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 plugin 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 open the (S)FTP preferences page, go to **Options > Preferences > (S)FTP**. In this preferences page you can customize the following options:

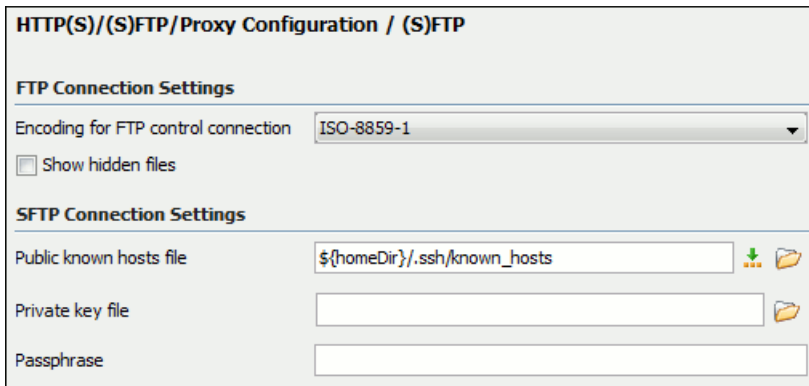


Figure 353: The (S)FTP Configuration Preferences Panel

- **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 plugin 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 file location 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.
- **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol.
- **Show SFTP certificate warning dialog** - If checked, a warning dialog will be shown each time when the authenticity of the host cannot be established.

Certificates Preferences

Oxygen XML Editor plugin provides two types of keystores for certificates 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, go to **Window > Preferences > oXygen XML Editor > XML > XML Signing Certificates**. You can customize the following parameters of a keystore:

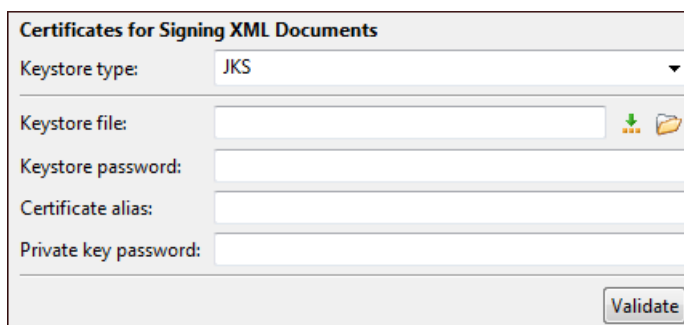


Figure 354: The Certificates Preferences Panel

- **Keystore type** - the type of keystore that Oxygen XML Editor plugin uses;
- **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 and / 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 Structure Outline Preferences

The **XML Structure Outline** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > XML Structure Outline** and contains the following preferences:

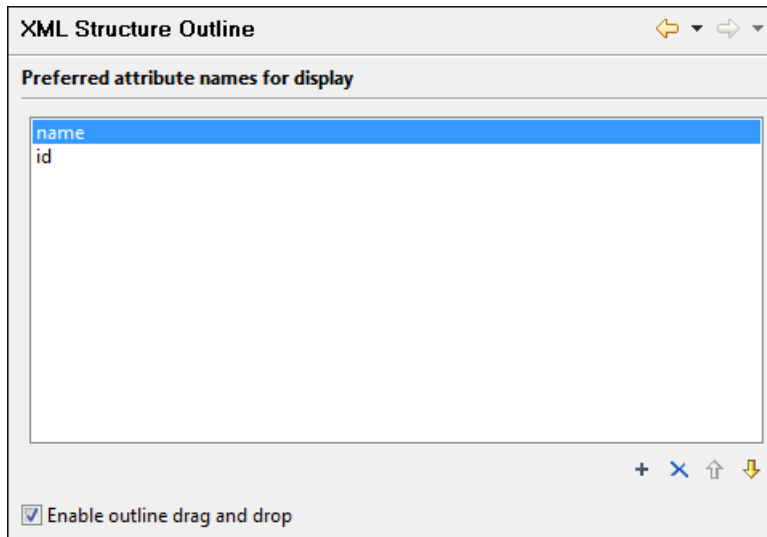


Figure 355: The XML Structure Outline Preferences Panel

- **Preferred attribute names for display** - The attribute names which should be preferred when displaying the element's attributes 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 should be disabled for the tree displayed in the **Outline** view only of there is a possibility to accidentally change the structure of the document by such drag and drop operations.

Scenarios Management Preferences

The **Scenarios Management** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > Scenarios Management** and allows sharing the global transformation scenarios with other users by exporting them to an external file that can be also imported in this preferences panel.

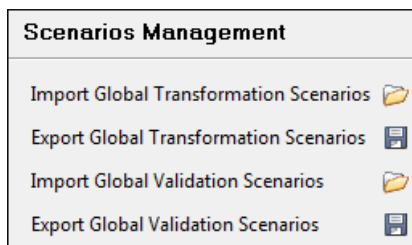


Figure 356: The Scenarios Management Preferences Panel

The actions available in this panel are the following:

- **Import Global Transformation Scenarios** - Allows you to import at global level all transformation scenarios from a properties file created with the export scenario action. The names of the imported scenarios will appear in the **Configure Transformation Scenario** dialog followed by **(import)**. This way there are no scenario name conflicts.

If you want to work with project level scenarios you have to first switch to project level in the **Configure Transformation Scenario** dialog.

- **Export Global Transformation Scenarios** - Allows you to export all global transformation scenarios available in the **Configure Transformation Scenario** dialog.
- **Import Global Validation Scenarios** - Allows you to import at global level all scenarios from a properties file created with the export scenario action. The names of the imported scenarios will appear in the **Configure Validation Scenario** dialog followed by (**import**). This way there are no scenario name conflicts.

If you want to work with project level scenarios you have to first switch to project level in the **Configure Validation Scenario** dialog.

- **Export Global Validation Scenarios** - Allows you to export all global validation scenarios available in the **Configure Validation Scenario** dialog.

Views Preferences

The **Views** preferences panel is opened from menu **Window > Preferences > oXygen XML Editor > Views** and contains the following preferences:

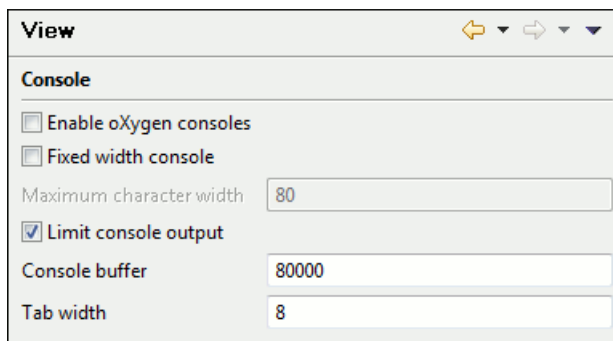


Figure 357: The Views Preferences Panel



- **Fixed width console** - If checked, a line in the **Console** view will be hard wrapped after the maximum numbers of characters allowed on a line.
- **Limit console output** - If checked, the content of the **Console** view will be limited to a configurable number of characters.
- **Console buffer** - Specifies the maximum number of characters that can be written in the **Console** view.
- **Tab width** - Specifies the number of spaces used for depicting a tab character.



Reset Global Options

To reset all global preferences to their default values you have to go to menu **Window > Preferences > oXygen XML Editor > Reset Global Options**. The list of transformation scenarios will be reset to the default scenarios.

Scenarios Management

You can import, export, and reset the global transformation and validation scenarios using the following actions:

- **Window > Preferences > oXygen / Scenarios Management >  Import Global Transformation Scenarios** - loads a set of transformation scenarios from a properties file that was created with the action **Export Global Transformation Scenarios**;
- **Window > Preferences > oXygen / Scenarios Management >  Export Global Transformation Scenarios** - stores all the global (not project-level) transformation scenarios in a properties file that can be used later by the action **Import Global Transformation Scenarios**;

- **Window > Preferences > oXygen / Scenarios Management >  Import Global Validation Scenarios** - loads a set of validation scenarios from a properties file that was created with the action **Export Global Validation Scenarios**;
- **Window > Preferences > oXygen / Scenarios Management >  Export Global Validation scenarios** - stores all the global (not project-level) validation scenarios in a separate properties file.

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, like 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, the command line of an external tool) to make a command or a parameter generic and reusable 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 plugin commands of external engines or other external tools, in transformation scenarios, validation scenarios, and Author operations:

- **`\${oxygenHome}** - Oxygen XML Editor plugin installation folder as URL;
- **`\${oxygenInstallDir}** - Oxygen XML Editor plugin installation folder as file path;
- **`\${frameworks}** - The path (as URL) of the `frameworks` subfolder of the Oxygen XML Editor plugin install folder;
- **`\${frameworksDir}** - The path (as file path) of the `frameworks` subfolder of the Oxygen XML Editor plugin installation folder;
- **`\${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;
- **`\${cfd}** - 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;
- **`\${cfu}** - The path of the current file as a URL. The current file is the one currently opened and selected;
- **`\${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, Mac 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; An 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. Can have one of the following values:
 - url - input is considered an URL. Oxygen XML Editor plugin checks that the URL is valid before passing it to the transformation;
 - password - input characters are hidden;
 - generic - the input is treated as generic text that requires no special handling;
 - relative_url - input is considered an URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing;



Note: You can use the `$ask` editor variable in file templates. In this case, Oxygen XML Editor plugin keeps an absolute URL.

- `combobox` - displays a dialog that contains a non-editable combo-box;
- `editable_combobox` - displays a dialog that contains an editable combo-box;
- `radio` - displays a dialog that contains radio buttons;
- 'default-value' - optional parameter. Provides a default value in the input text box;

Examples:

- ``${ask('message')}`` - Only the message displayed for the user is specified.
- ``${ask('message', generic, 'default')}`` - 'message' is displayed, the type is not specified (the default is string), the default value is 'default'.
- ``${ask('message', password)}`` - 'message' is displayed, the characters typed are masked with a circle symbol.
- ``${ask('message', password, 'default')}`` - same as before, the default value is 'default'.
- ``${ask('message', url)}`` - 'message' is displayed, the parameter type is URL.
- ``${ask('message', url, 'default')}`` - same as before, the default value is 'default'.
- **`\${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.

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 a , 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 [Preferences page](#).

Localization of the User Interface

To localize the Oxygen XML Editor plugin Eclipse plugin, you can use one of the following methods:

- localization through the update site:

Start Eclipse, go to **Help > Install New Software...** Press Add Site in the **Available Software** tab of the **Software Updates** dialog. Enter <http://www.oxygenxml.com/InstData/Editor/Eclipse/site.xml> in the location field of the **Add Site** dialog. Press OK. Select the language pack checkbox.

- localization through the zip archive:

Go to <http://www.oxygenxml.com/download.html> and download the zip archive with the plugin language pack. Unzip the downloaded zip archive in the dropins subdirectory of the Eclipse install directory. Restart Eclipse.

If your operating system is running in the language you want to start Eclipse in (for example, you are using Japanese version of Windows XP, and you want to start Eclipse in Japanese), Oxygen XML Editor plugin matches the appropriate language from the language pack. However, if your operating system is running in a language other than the one you want to start Eclipse in (for example, you are using the English version of Windows XP, and you want to start Eclipse in Japanese, if you have the required operating system language support including the keyboard layouts and input method editors installed), specify the `-nl <locale>` command line argument when you launch Eclipse. Oxygen XML Editor plugin uses the translation file which matches the specified `<locale>`.

You can also localize the Eclipse plugin to a different language than the initial languages in the language pack. Duplicate the `plugin.properties` file from the Oxygen XML Editor plugin installation directory, translate all the keys in the file and change its name to `plugin_<locale>.properties`.

Chapter 20

Upgrading Oxygen XML Editor plugin

Topics:

- [Upgrading Oxygen XML Editor plugin on Windows / Linux](#)
- [Upgrading Oxygen XML Editor plugin on Mac OS X](#)

This chapter presents the procedure you have to follow to upgrade Oxygen XML Editor plugin.

Upgrading Oxygen XML Editor plugin on Windows / Linux

The following steps describe the upgrading procedure of Oxygen XML Editor plugin, valid for Windows:

1. In case you added files, or modified files from the installation folder of Oxygen XML Editor plugin, save a copy of them. These files are overwritten during the upgrade.



Note: Custom frameworks are preserved, but we recommend you to back them up as well.

2. Run the new installation kit and install the new version in the folder where the old version was installed.
3. When you are promoted to choose whether you want to upgrade the existed installation, select **Proceed**. The old version is uninstalled automatically and the new one takes its place.
4. Copy the files you previously backed up in the installation folder.

Upgrading Oxygen XML Editor plugin on Mac OS X

The following steps describe the upgrading procedure of Oxygen XML Editor plugin, valid for Mac OS X:

1. Back up anything you need from the installation folder of Oxygen XML Editor plugin.
2. Remove the installation folder.
3. Unpack the archive with the new version.



Note: Do not unpack a new version of Oxygen XML Editor plugin on top of an older version. This causes problems because of used libraries that become duplicated.

4. Copy the backed-up files in the new installation folder.

Chapter 21

Common Problems

Topics:

- [*Oxygen XML Editor plugin Takes Several Minutes to Start on Mac*](#)
- [*XSLT Debugger Is Very Slow*](#)
- [*Syntax Highlight Not Available in Eclipse Plugin*](#)
- [*Problem Report Submitted on the Technical Support Form*](#)
- [*Signature verification failed error on open or edit a resource from Documentum*](#)
- [*Compatibility Issue Between Java and Certain Graphics Card Drivers*](#)
- [*An Image Appears Stretched Out in the PDF Output*](#)
- [*The DITA PDF Transformation Fails*](#)
- [*Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x*](#)
- [*JPEG CMYK Color Space Issues*](#)
- [*MSXML 4.0 Transformation Issues*](#)

This chapter presents common problems that may appear when running the application and the solutions for these problems.

Oxygen XML Editor plugin Takes Several Minutes to Start on Mac

If Oxygen XML Editor plugin 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**.

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).

Syntax Highlight Not Available in Eclipse Plugin

I associated the `.ext` extension with Oxygen XML Editor plugin in Eclipse. Why does an `.ext` file opened with the Oxygen XML Editor plugin not have syntax highlight?

Associating an extension with Oxygen XML Editor plugin in Eclipse 3.7+ requires three steps:

1. Associate the `.ext` extension with the Oxygen XML Editor plugin.
 - a) Go to menu **Windows** > **Preferences** > **General** > **Editors** > **File Associations**.
 - b) Add `*.ext` to the list of file types.
 - c) Select `*.ext` in the list by clicking on it.
 - d) Add Oxygen XML Editor plugin to the list of **Associated editors** and make it the default editor.
2. Associate the `.ext` extension with the **Oxygen XML** content type.
 - a) Go to menu **Windows** > **Preferences** > **General** > **Content Types**.
 - b) Add `*.ext` to the **File associations** list for the **Text** > **XML** > **oXygen XML** content type.
3. Press the **OK** button in the Eclipse preferences dialog.

Now when an `*.ext` file is opened the icon of the editor and the syntax highlight should be the same as for XML files opened with the Oxygen XML Editor plugin.

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

Oxygen XML Editor plugin | Signature verification failed error on open or edit a resource from Documentum | 761
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.

Edit the `eclipse.ini` file from the Eclipse directory and add the following parameter to the `-vmargs`:

`-Drequire.signed.ucf.jars=false`, for example:

```
-vmargs
-Xms40m
-Xmx256m
-Drequire.signed.ucf.jars=false
```

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 to go to **Preferences > 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 than ON.

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.

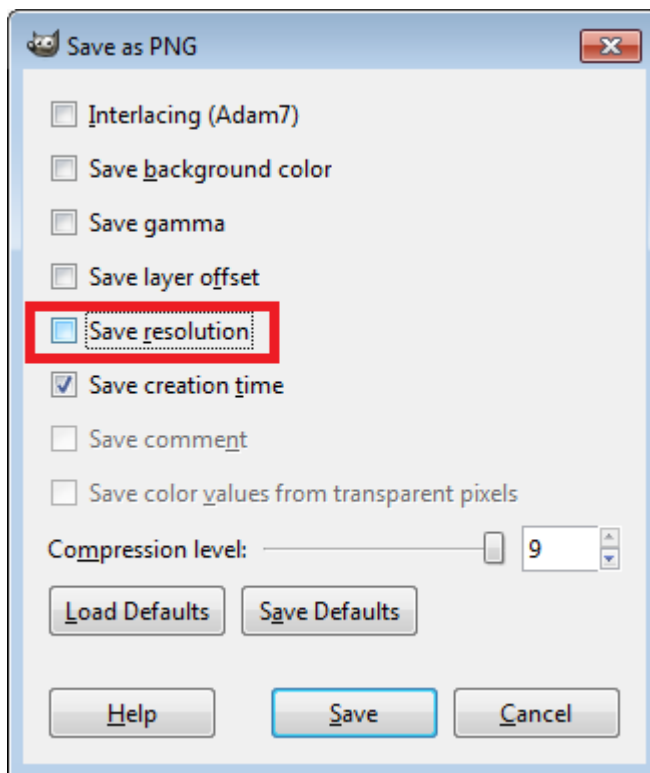
The PDF processor which comes out of the box with the application is the open-source Apache FOP processor. Here is what Apache FOP does when deciding the image size:

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_INSTALL_DIR\lib\fop.xconf`. DITA publishing uses the DITA Open Toolkit which has the Apache FOP configuration file located in `OXYGEN_INSTALL_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 uses the DITA Open Toolkit.

In case 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 emphasises errors like topics referenced in other topics but not in the DITA Map, broken links, and missing external resources.

You can also analyse the **results** tab of the DITA transformation and search for messages that contain text similar to `[fop] [ERROR]`. In case you encounter this type of error messages, 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 the Oxygen XML team.

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 plugin 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 plugin, go to the installation directory of Oxygen XML Editor plugin and rename or move the `jre` folder. If Oxygen XML Editor plugin 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_INSTALL_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.

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.

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

Chapter

22

Using the WebApp Reviewer

Topics:

- [The Review Mode](#)
- [The Edit Mode](#)

The WebApp Reviewer is a browser based application for mobile devices, especially designed to help you review and edit DITA documents, offering a simplified version of Oxygen XML Editor plugin's **Author** mode functionality. Portability and ease of use are the main advantages of the WebApp Reviewer. The most common authoring actions are available in tow working modes:

- [The Review mode](#);
- [The Edit mode](#).

The Review Mode


The **Review** mode in the WebApp Reviewer provides an intuitive interface that enables you to view documents and document edits. Swipe right to display the **Review** panel at the left of the editing area. This panel presents comments and tracked changes. A **Preview** panel is also available at the bottom of the editing area. It displays the current comment or tracked change and the author who inserted it..

To insert a comment, select the text that you want to comment on and touch **Add comment** at the bottom of the editing area. You are able to navigate between comments using the two arrows (left and right) from the **Preview** panel. To make changes to a comment, touch **Edit**.

The Edit Mode

The **Edit** mode in the WebApp Reviewer allows you not to only review a document but to also make changes to it. You also have the possibility to enable the **Track Changes** support and record all the edits you make.

When you swipe right, the **Toolbar** panel is displayed at the left of the editing area. This panel lets you enable the track changes support, display the content completion list of proposals, undo/redo a change, and rename an element.

 **Important:** On Android devices the content completion list of proposals might display *undefined* elements. To avoid this, go to **Settings > Bandwidth Management > Reduce data usage** and select **OFF**.

Index

A

- Archives 621, 622, 624
- browse 622
- edit 624
- file browser 622
- modify 622
- Author Editor 62, 63, 64, 65, 68, 70, 71, 72, 73, 74, 76, 79, 88, 89, 90, 91, 93, 94, 95, 96, 97, 99
- attributes view 68
- breadcrumb 71
 - change tracking 90, 91, 93, 94, 95, 96, 97, 99
 - callouts 97
 - manage changes 90
 - managing comments 95, 96
 - the Review view 99
 - track changes behavior 91
 - track changes limitations 93
- content author role 64
- contextual menu 74
- edit content 79
- editing XML 76
- edit markup 76
- elements view 68
- entities view 70
- external references 74
 - navigation 71, 72
 - bookmarks 72
 - display the markup 72
- outline view 65
- position information tooltip 73
- reload content 88
- roles: content author, framework developer 63
- validation 88
 - whitespace handling 89, 90
 - versions differences 90
- WYSIWYG editing 63
- Author Settings 401, 403, 404, 405, 406, 407, 412, 413, 415, 434, 437, 439, 444, 447, 448, 451, 453, 455
 - actions 401, 403
 - insert section 401
 - insert table 403
- Author default operations 407
 - content 406
 - configuring the content completion 406
 - content completion customization wizard 406
 - Java API 412, 415, 434, 437, 439, 444, 447, 448, 451, 453, 455
 - Author extension state listener 437
 - Author schema aware editing handler 439
 - configure XML node renderer customizer 455
 - CSS styles filter 447
 - customize outline icons 455
 - customize XML node 455
 - extensions bundle 434
 - generate unique ID 455
 - references resolver 444
 - table cell row and column separators provider 453

- Author Settings (*continued*)
 - Java API (*continued*)
 - table cell span provider 451
 - table column width provider 448
- Java API example 413
 - menus 401, 404, 405
 - contextual menu 405
 - main menu 404
 - toolbars 401, 404
 - configure toolbar 404

B

- Bidirectional text 62, 106
- Author Mode 106
- Grid Mode 62

C

- Common Problems 759
- Configure the Application 419, 428, 692, 693, 694, 703, 705, 706, 707, 708, 709, 711, 712, 713, 714, 715, 716, 717, 719, 720, 721, 723, 724, 725, 726, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 740, 741, 742, 743, 746, 747, 748, 749, 750, 751, 752, 753
- (S)FTP 749
- archive 747
- certificates 750
- CSS validator 725
- customize default options 692
- custom validation 724
 - data sources 743, 746, 747
 - download links for database drivers 746
 - table filters 747
- document type association 694
- editor preferences 705
 - Editor preferences 705, 706, 707, 708, 709, 711, 712, 713, 714, 715, 716, 717, 719, 720, 721, 723
 - author 708
 - author profiling conditional text 713
 - author track changes 711
 - callouts 712
 - code templates 721
 - content completion 717
 - document checking 723
 - document templates 721
 - elements and attributes by prefix 720
 - format 714
 - format - CSS 716
 - format - JavaScript 716
 - format - XML 715
 - grid 707
 - open/save 720
 - pages 706
 - print 705
 - schema aware 709
 - schema design 713
 - spell check 721

- Configure the Application (*continued*)
 - Editor preferences (*continued*)
 - syntax highlight 719
 - text/diagram 707, 714
- editor variables 428, 753
- fonts 694
- HTTP(S)/WebDAV preferences 749
 - import 741, 742
 - date/time patterns 742
- import/export global options 692
- internationalization 419, 703
- license 694
- outline 751
- reset global options 752
- scenarios management 751, 752
- views 752
- XML 725
- XML catalog 725
- XML instances generator 728
- XML parser 726
- XProc engines 729
- XSLT 730
- XSLT/FO/XQuery 730
 - XSLT/FO/XQuery preferences 730, 731, 732, 733, 734, 735, 736, 737, 738, 740
 - custom engines 740
 - debugger 737
 - FO Processors 738
 - MSXML 734
 - MSXML.NET 734
 - profiler 737
 - Saxon6 730
 - Saxon HE/PE/EE 731, 736
 - Saxon-HE/PE/EE 731
 - Saxon HE/PE/EE advanced options 732, 736
 - XPath 740
 - XQuery 735
 - XSLTProc 733
- Content Management System 677
- Content Reuse 326, 327, 328
- content references 326
- insert a direct content reference 328
- reusable components 327
- Copy/Paste 60, 106
- grid editor 60
- smart paste 106
- CSS arithmetic functions 493
- CSS arithmetic extensions 493
- CSS Support in <oxygen/> Author 458, 462, 469
 - CSS 2.1 features 458, 462
 - properties support table 462
 - supported selectors 458
 - Oxygen CSS extensions 458, 469
 - media type oxygen 458
- Customization Support 395, 398, 399, 420, 427, 430, 431, 433, 441, 496
 - document type associations (advanced customization tutorial) 399, 420, 427, 430, 431, 433, 441
 - Author settings 399
 - basic association 420
 - configuring extensions - link target reference finder 441
 - configuring transformation scenarios 431

- Customization Support (*continued*)
 - document type associations (advanced customization tutorial) (*continued*)
 - configuring validation scenarios 433
 - new file templates 427
 - XML Catalogs 430
 - example files 496
 - the Simple Documentation Framework Files 496
 - simple customization tutorial 395, 398
 - CSS 395
 - XML instance template 398
 - XML Schema 395

D

- Databases 594, 625, 626, 645, 646, 660, 662, 663, 683
- debugging with MarkLogic 662, 663
- limitations of the MarkLogic debugger 663
- native XML databases (NXD) 645
- Native XML databases (NXD) 646
- Relational databases 626
- SharePoint connection 683
- WebDAV connection 663
 - XQuery 594, 660, 662
 - debugging 662
 - drag and drop from the Data Source Explorer 660
 - transformation 660
 - validation 594
- Debugging XSLT/XQuery Documents 600, 603, 611, 614
- Java extensions 614
 - layout 600, 603, 611
 - information views 603
 - multiple output documents in XSLT 2.0 611
- XSLT/XQuery debugger 611
- Debugging XSLT / XQuery Documents 601
 - layout 601
 - Control toolbar 601
- Digital Signature 686, 687, 688, 689
 - canonicalizing files 687
 - certificates 688
 - signing files 688
 - verifying the signature 689
- DITA MAP document type 372
 - association rules 372
 - Author extension 372
 - catalogs 372
 - schema 372
- DITA MAP Document Type 372, 373, 382
 - Author extension 372, 373, 382
 - templates 382
 - transformation scenarios 373
- DITA Maps 298, 300, 301, 303, 304, 305, 307, 308, 317, 320, 321, 322, 324, 325
 - advanced operations 305, 307, 308
 - edit properties 308
 - inserting a topic group 307
 - inserting a topic heading 307
 - inserting a topic reference 305
- creating a Bookmap 304
- creating a DITA Map 301
- creating a subject scheme 304
- creating a topic 303

DITA Maps (*continued*)

- DITA OT customization support 320, 321
- customizing the <oXygen/> Ant tool 320
- increase the memory for Ant 320
- resolve topic reference through an XML catalog 321
- use your own custom build file 320

DITA OT installation plugin 322

- DITA specialization 325
- editing DITA Map specialization 325
- DITA specialization support 324, 325
- editing DITA Topic specialization 325

DITA transformation scenario 308

editing actions 300

organizing topics 303

relationships between topics 304

- transforming DITA Maps 308, 317
- running an ANT transformation 317

validating a DITA Map 301

DITA Topics document type 364

association rules 364

- Author extensions 364
- catalogs 364

schema 364

DITA Topics Document Type 364, 371

- Author extensions 364, 371
- templates 371
- transformation scenarios 371

DITA transformation scenario 308, 311

customize scenario 311

DocBook Targetset document type 363

association rules 363

schema 363

DocBook Targetset Document Type 363, 364

- Author extensions 364
- templates 364

DocBook V4 document type 336, 347

association rules 336

- Author extensions 336, 347
- catalogs 336
- templates 347

schema 336

DocBook V4 Document Type 336, 339

- Author extensions 336, 339
- transformation scenarios 339

DocBook V5 document type 350, 361

association rules 350

- Author extensions 350, 361
- catalogs 350
- templates 361

schema 350

DocBook V5 Document Type 350, 351

- Author extensions 350, 351
- transformation scenarios 351

Documentum (CMS) Support 678, 679, 680, 681

- actions 679, 680, 681
- cabinets/folders 680
- connection 680
- resources 681

configuring a Documentum (CMS) data source 678, 679

E

Edit 101, 109, 110, 115, 119, 289, 293, 294, 624

- archives 624
- associating a file extension 294
- check spelling 289
- check spelling in files 293
- close documents 119
- conditional text 101
- create new documents 110
- file properties 119
- open and close documents 110
- open read-only files 294
- open remote documents (FTP/SFTP/WebDAV) 115
- open the currently document in the system application 119
- save documents 115
- Unicode documents 110
- Unicode support 110

Editing CSS Stylesheets 260, 261, 262

Content Completion Assistant 260

folding 262

format and indent (pretty print) 262

other editing actions 262

Outline view 261

validation 260

Editing JavaScript Documents 279

Editing JavaScript Files 279, 280, 281, 282

Content Completion Assistant 280

Outline view 281

Text mode 279

validating JavaScript files 282

Editing JSON Documents 274, 275, 276, 277

- convert XML to JSON 277
- folding 275
- Grid mode 276
- Outline view 277
- syntax highlight 275
- Text mode 274

Validating JSON Documents 277

Editing NVDL Schemas 271, 272, 273

Component Dependencies view 273

editor specific actions 273

- schema diagram 271, 272, 273
- actions in the diagram view 272
- full model view 271
- Outline view 273

searching and refactoring actions 273

Editing RelaxNG Schemas 269

Component Dependencies View 269

Editing Relax NG Schemas 262, 263, 264, 265, 266, 267

editor specific actions 266

Resource Hierarchy/Dependencies View 267

- schema diagram 262, 263, 264, 265
- actions 265
- full model view 263
- logical model view 264
- Outline view 265
- symbols 263

searching and refactoring actions 267

Editing Schematron Documents 288

searching and refactoring operations 288

- Editing Schematron Schemas 283, 284, 286
 - contextual editing 286
 - validation against Schematron 284
- Editing StratML Documents 278
- Editing WSDL Document 257
 - SOAP request 257
 - composing a SOAP request 257
- Editing WSDL Documents 244, 245, 248, 249, 250, 252, 253, 254, 257, 259
- Component Dependencies view 252
- component occurrences 253
- composing web service calls with WSDL SOAP analyzer 257
- content completion 248
- contextual editing 248
- generate documentation for WSDL documents 254
- generate documentation for WSDL documents from command line 257
- generate documentation for WSDL documents in a custom format 257
- Outline view 245
- Quick Assist 253
- Resource Hierarchy/Dependencies view 250
- searching and refactoring operations 249
- searching and refactoring operations scope 249
 - SOAP request 259
 - testing remote WSDL files 259
 - UDDI registry browser 259
- Editing XML Documents 78, 120, 124, 125, 127, 131, 133, 134, 136, 137, 138, 139, 140, 143, 144, 145, 148, 150, 151, 153, 155, 159, 160, 161, 162
 - against a schema 137
 - associate a schema to a document 124, 125, 127
 - add schema association in XML instance 125
 - learning a document structure 127
 - setting a default schema 124
 - supported schema types 124
 - checking XML well-formedness 136
 - converting between schema languages 153
 - document navigation 144, 145
 - folding 144
 - outline view 145
 - editor specific actions 159, 160, 161, 162
 - document actions 160
 - edit actions 159
 - refactoring actions 161
 - select actions 160
 - smart editing 162
 - source actions 160
 - syntax highlight depending on namespace prefix 162
- formatting and indenting documents (pretty print) 155
 - grouping documents in XML projects 120, 148
 - large documents 148
 - new project 120
 - project view 120
- including document parts with XInclude 148
- Resource Hierarchy/Dependencies view 151
- status information 159
 - streamline with content completion 78, 127, 131, 133, 134
 - code templates 78, 134
 - the Annotation panel 133
 - the Attributes view 133
 - the Elements view 133

- Editing XML Documents (*continued*)
 - streamline with content completion (*continued*)
 - the Entities view 134
 - the Model panel 131
 - validation against a schema 137, 138, 139, 140, 143, 144
 - automatic validation 138
 - custom validation 139
 - marking validation errors 137
 - resolving references to remote schemas with an XML Catalog 144
 - validation actions 143
 - validation example 138
 - validation scenario 140
- working with XML Catalogs 150
- Editing XML Schemas 187, 221, 225, 226, 228, 231, 232, 233, 235, 237
- Component Dependencies view 237
- contextual editing 187
 - generate documentation for XML Schema 226, 228, 231, 232
 - as HTML 228
 - as PDF, DocBook or custom format 231
 - from command line 232
- relational database table to XML schema 221
- Resource Hierarchy/Dependencies view 235
- schema instance generator 221
- schema regular expressions builder 225
- searching and refactoring actions 233
- Editing XProc Scripts 282
- Editing XQuery Documents 241, 242, 243
- folding 242
- generate HTML documentation 243
- Editing XSL Stylesheets 184, 185
 - Component Dependencies view 184
 - quick assist support 185
 - XSpec 185
- Editing XSLT Schemas 165
 - contextual editing 165
- Editing XSLT Stylesheets 164, 165, 166, 170, 171, 173, 174, 176, 179, 180, 181, 182
 - content completion 165, 166, 170
 - code templates 170
 - in XPath expressions 166
- find XSLT references and declarations 180
- generate documentation for XSLT stylesheets 174
 - generate documentation for XSLT Stylesheets 176, 179
 - as HTML 176
 - from command line 179
 - in custom format 179
- Outline view 171
- refactoring actions 181
- Resource Hierarchy/Dependencies view 182
 - validation 164
 - custom validation 164
 - validation scenario 164
- XSLT Input view 170
- XSLT stylesheet documentation 173
- EPUB Document Type 392

F

- Find/Replace 56
- Find All Elements/Attributes dialog 56

G

Getting Started 43, 44, 45, 49, 51, 52
 help 44

- perspectives 44, 45, 49, 51, 52
 - database 52
 - editor 45
 - XQuery debugger 51
 - XSLT debugger 49

 supported types of documents 44
 grid editor 59

- navigation 59
 - collapse all 59
 - collapse children 59
 - collapse others 59
 - expand all 59
 - expand children 59

 Grid Editor 57, 58, 59, 60

- add nodes 60
- clear column content 59
- copy/paste 60
- drag and drop 60
- duplicate nodes 60
- inserting table column 59
- insert table row 59
- layouts (grid and tree) 58
- navigation 58
- refresh layout 60
- sort table column 59
- start editing a cell value 60
- stop editing a cell value 60

I

Importing data 668
 Importing Data 668, 671, 674

- from a database 668
 - table content as XML document 668
- from database 671
 - convert table structure to XML Schema 671
- from HTML files 674
- from MS Excel 671
- from text files 674

 Installation 30, 31

- Eclipse 31
 - update site method (Eclipse 3.4 - 4.2) 31
 - ZIP archive method (Eclipse 3.4 - 4.2) 31

 environment 30
 operating system requirements 30
 platform requirements 30

L

License 32, 34, 36, 39
 floating (concurrent) license 34
 floating license server 36
 floating license servlet 34
 license server installed as Windows service 36
 register a license key 32
 registration code 39
 release floating license 39
 unregister license key 39

M

Master Files 122

N

Native XML Databases (NXD) 637, 644, 645, 646, 647, 648

- database connections configuration 646, 647, 648
 - Berkeley DB XML 646
 - Documentum xDb (X-Hive/DB) 648
- eXist 647
 - data sources configuration 644, 645, 646
 - Berkeley DB XML 645
 - Documentum xDb (X-Hive/DB) 646
 - eXist 645
 - MarkLogic 646
 - resource management 637, 648
 - Data Source Explorer view 637, 648

O

OutOfMemory 41, 714, 720, 738
 Out Of Memory 41, 313, 556, 559, 714, 720, 738
 memory allocated for a transformation 313, 556
 OutOfMemoryError 41, 714, 720, 738
 Oxygen CSS extensions 475, 478

- <oXygen/> CSS custom functions 475, 478
 - oxy_unparsed-entity-uri()* 478
 - oxy_url()* 475

 Oxygen CSS Extensions 460, 461, 466, 469, 471, 472, 473, 474, 475

- <oXygen/> CSS custom functions 475
 - additional properties 471, 472, 473, 474
 - display tags 474
 - editable property 472
 - folding elements 471
 - link elements 473
 - morph value 472
 - placeholders for empty elements 472
 - supported features from CSS level 3 460, 466, 469
 - additional custom selectors 469
 - attr() function 466
 - namespace selectors 460
 - supported features from CSS level 4 461
 - subject selectors 461

 Oxygen Tools 685

P

Performance Problems 41

- external processes 41

 Profile DITA step by step 333

- conditional text 333
- profiling tutorial 333

 Profiling 330, 331, 618

- conditional text 331
- filter content 331
- filter content 330
- conditional text 330

 XSLT stylesheets and XQuery documents 618

Profiling XSLT Stylesheets and XQuery Documents 618, 619
 profiling information 618, 619
 Hotspots view 619
 Invocation tree view 618
 XSLT/XQuery profiler 619

Q

Querying Documents 241, 586, 589, 591, 592, 593, 594, 595
 running XPath and XQuery expressions 589
 XPath/Xquery Builder view 589
 running XPath expressions 586
 XPath dialog 586
 XQuery 241, 591, 592, 593, 594, 595
 Input view 593
 other editing actions 595
 Outline view 241, 592
 syntax highlight and content completion 591
 transforming XML documents; advanced Saxon B/SA options
 595
 validation 594

R

Relational Databases 626, 627, 628, 629, 630, 632, 633, 634, 635,
 636, 637, 640, 642, 644, 648, 668, 671
 connections configuration 632, 633, 634, 635, 636
 generic JDBC 635
 IBM DB2 connection 632
 JDBC-ODBC connection 633
 Microsoft SQL Server 633
 MySQL 634
 Oracle 11g 635
 PostgreSQL 8.3 636
 creating XML Schema from databases 671
 data sources configuration 626, 627, 628, 629, 630
 generic JDBC data source 628
 IBM DB2 626
 Microsoft SQL Server 627
 MySQL 628
 Oracle 11g 629
 PostgreSQL 8.3 630
 importing from databases 668
 resource management 637, 640, 648
 Data Source Explorer view 637, 648
 Table Explorer view 640
 SQL execution support 642, 644
 drag and drop from the Data Source Explorer 642
 executing SQL statements 644
 SQL validation 644
 Relax NG Schema Editor 262
 contextual editing 262

S

SharePoint Connection 683, 684
 actions at connection level 683
 actions at file level 684
 actions at folder level 684
 configuration 683

T

TEI ODD document type 385
 association rules 385
 Author extensions 385
 catalogs 385
 schema 385
 TEI ODD Document Type 385, 387
 Author extensions 385, 387
 templates 387
 transformation scenarios 387
 TEI P4 document type 387, 388
 association rules 387
 Author extensions 388
 catalogs 388
 schema 387
 TEI P4 Document Type 387, 388, 389, 390
 Author extensions 388, 389, 390
 templates 390
 transformation scenarios 389
 TEI P5 document type 390
 association rules 390
 schema 390
 TEI P5 Document Type 390, 391
 Author extensions 391
 templates 391
 transformation scenarios 391
 Text Editing Mode 56
 Transformation Scenario 545, 546, 550, 551, 568, 570, 571
 batch transformation 571
 built-in transformation scenarios 571
 new transformation scenario 545, 546, 550, 551, 568, 570
 additional XSLT stylesheets 551
 configure transformation scenario 545
 create a transformation scenario 570
 XML transformation with XSLT 546
 XSLT/XQuery extensions 568
 XSLT parameters 550
 Transforming Documents 339, 341, 543, 544, 545, 571, 578, 579,
 580
 custom XSLT processors 579
 output formats 339, 341, 544
 WebHelp 339
 WebHelp with feedback 341
 supported XSLT processors 578
 transformation scenario 545
 Transformation Scenarios view 571
 XSL-FO processors 580
 XSLT processors extensions paths 579

U

Uninstalling the Plugin 40
 Upgrade 39, 40
 check for new version 40

V

Validating XML Documents 136

W

- WebDAV Connection 663, 664, 665
- actions at connection level 664
- actions at file level 665
- actions at folder level 664
- configuration 664
- WebHelp Internationalization 377
 - WebHelp localization 377
 - WebHelp i18n 377
- WebHelp Systems 373
- WebHelp Systems with Feedback 377

X

- XHTML document type 383
- association rules 383
 - Author extensions 383
 - catalogs 383
- CSS 383
- schema 383
- XHTML Document Type 383, 384, 385
 - Author extensions 383, 384, 385
 - templates 385
 - transformation scenarios 384
- XML Outline View 66, 67, 145, 146, 147
 - Author 67
 - outline filters 67
- contextual menu 67
 - document structure change 66, 147
 - popup menu 147
- document tag selection 147
- modification follow-up 66, 147
- outliner filters 146
- XML document overview 66, 146
- XML Schema Diagram Editor 190, 192, 193, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 217, 218, 219, 220, 221
- Attributes view 218
- editing actions 211
- edit schema namespaces 221
 - Facets view 219, 220
 - editing patterns 220
 - group schema components 209
 - attributes 209
 - constraints 209
 - substitutions 209
- navigation 210
- Outline view 217
 - schema components 192, 193, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208
 - xs:alternative 201
 - xs:any 204

XML Schema Diagram Editor (*continued*)

- schema components (*continued*)
 - xs:anyAttribute 205
 - xs:assert 207
 - xs:attribute 196
 - xs:attributeGroup 197
 - xs:complexType 198
 - xs:element 193
 - xs:field 207
 - xs:group 202
 - xs:import 202
 - xs:include 202
 - xs:key 206
 - xs:keyRef 206
 - xs:notation 203
 - xs:openContent 208
 - xs:override 203
 - xs:redefine 203
 - xs:schema 192
 - xs:selector 207
 - xs:sequence, xs:choice, xs:all 204
 - xs:simpleType 199
 - xs:unique 206
- the Palette view 220
- validation 210
- XML Schema Text Editor 187, 188
- content completion 187
- flatten an XML Schema 188
- XQJ Connection 596
- XQJ configuration 596
- XQJ Support 596
- XQJ processor configuration 596
- XSLT/XQuery Debugger 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613
 - debug steps 612
 - determining what XSLT/XQuery expression generated particular output 613
 - using breakpoints 612, 613
 - inserting breakpoints 612
 - removing breakpoints 613
 - viewing processing information 603, 604, 605, 606, 607, 608, 609, 610
 - breakpoints view 604
 - context node view 603
 - messages view 605
 - node set view 609
 - output mapping stack view 607
 - stack view 606
 - templates view 608
 - trace history view 607
 - variables view 610
 - XPath watch view 603

