Oxygen XML Editor 12.2

Contents

Chapter 1: Introduction....17

Key Features and Benefits of Oxygen XML Editor 18

Chapter 2: Installation....21

Installation Requirements....22
- Platform Requirements....22
- Operating System....22
- Environment Requirements....22
- JWS-specific Requirements....22

Installation Instructions....23
- Windows Installation....23
- Mac OS X Installation....24
- Linux Installation....24
- All Platforms Installation....25
- Unix / Linux Server Configuration....25
- Windows NT Terminal Server Configuration....26
- Unattended Installation....26

Java Web Start (JWS) Installer....27

Setting a Parameter in the Launcher Configuration File / Startup Script....29

Starting the Application....30
- Starting the Application on Windows....30
- Starting the Application on Mac OS X....30
- Starting the Application on Linux....30
- Starting the Application with the All Platforms Kit....30

Obtaining and Registering a License Key....31
- Named User License Registration....31
- Named User License Registration with Text File....32
- How Floating (Concurrent) Licenses Work....32
- License Registration with an Activation Code....37

Unregistering the License Key....37

Upgrading the Oxygen Application....37
- Upgrading the Standalone Application....37

Checking for New Versions....38

Uninstalling the Application....38
- Uninstalling the Standalone Application....38
- Unattended Uninstall....38

Performance Problems....38
- Large Documents....39

External Processes....39
Display Problems on Linux/Solaris....39

Chapter 3: Getting Started....41

Getting Help....42
Supported Types of Documents....42
Perspectives....42
Editor Perspective....43
XSLT Debugger Perspective....44
XQuery Debugger Perspective....45
Database Perspective....45
Tree Editor Perspective....47
Dockable Views and Editors....48

Chapter 4: Editing Documents....51

Working with Unicode....52
Opening and Saving Unicode Documents....52
The Unicode Toolbar....53
Opening and Closing Documents....53
Creating New Documents....54
Saving Documents....57
Opening Existing Documents....58
Opening and Saving Remote Documents via FTP/SFTP/WebDAV....59
Opening the Current Document in a Web Browser....63
Closing Documents....63
Viewing File Properties....63
Editing XML Documents....63
Associate a Schema to a Document....64
Streamline with Content Completion....67
Validating XML Documents....74
Document Navigation....83
Grouping Documents in XML Projects....87
Working with XML Catalogs....93
Converting Between Schema Languages....94
Editing XML Tree Nodes....97
Formatting and Indenting Documents (Pretty Print)....97
Viewing Status Information....98
Image Preview....98
Making a Persistent Copy of Results....98
Locking and Unlocking XML Markup....99
Adjusting the Transparency of XML Markup....99
XML Editor Specific Actions....99
Editing XML Schemas....103

XML Schema Text Editor....103
XML Schema Diagram Editor....110
Contextual Editing....138
Create an XML Schema From a Relational Database Table....138
Generate Sample XML Files....138
XML Schema Regular Expressions Builder....143
Generating Documentation for an XML Schema....145
Searching and Refactoring Actions....152
Search and Refactor Operations Scope....154
Resource Hierarchy / Dependencies View....155
Component Dependencies View....157
Highlight Component Occurrences....158
XML Schema Quick Assist Support....158
Linking Between Development and Authoring....159
Editing Relax NG Schemas....159
Relax NG Schema Diagram....159
Relax NG Editor Specific Actions....163
Searching and Refactoring Actions....164
Resource Hierarchy/Dependencies View....164
Component Dependencies View....166
RNG Quick Assist Support....166
Configuring a Custom Datatype Library for a RELAX NG Schema....167
Linking Between Development and Authoring....167
Editing NVDL Schemas....167
NVDL Schema Diagram....168
NVDL Editor Specific Actions....169
Searching and Refactoring Actions....169
Component Dependencies View....170
Linking Between Development and Authoring....170
Editing XSLT Stylesheets....170
Validating XSLT Stylesheets....171
Contextual Editing....171
Content Completion in XSLT Stylesheets....171
The XSLT/XQuery Input View....176
The XSLT Outline View....178
XSLT Stylesheet Documentation Support....179
Generating Documentation for an XSLT Stylesheet....180
Finding XSLT References and Declarations....188
Highlight Component Occurrences....188
XSLT Refactoring Actions....188
Resource Hierarchy/Dependencies View....189
Component Dependencies View....190
XSLT Quick Assist Support....191
Linking Between Development and Authoring....192
Editing XQuery Documents....192

XQuery Outline View....192
Folding in XQuery Documents....194
Generating HTML Documentation for an XQuery Document....194
Editing CSS Stylesheets....195
Validating CSS Stylesheets....195
Content Completion in CSS Stylesheets....196
CSS Outline View....196
Folding in CSS Stylesheets....197
Formatting and Indenting CSS Stylesheets (Pretty Print)....197
Other CSS Editing Actions....197
Editing XProc Scripts....197
Editing Schematron Schemas....198
Combined RELAX NG / W3C XML Schemas and Schematron Schema....198
Validate an XML Document....199
SVG Documents....199
The Standalone SVG Viewer....200
The Preview Result Panel....200
Integrating External Tools....201
Editing Very Large Documents....202
Insufficient Memory....202
Large File Viewer....202
Hex Viewer....204
Scratch Buffer....204
Changing the User Interface Language....204
Handling Read-Only Files....205
Editing Documents with Long Lines....205

Chapter 5: Authoring in the Tagless Editor....207

Authoring XML Documents Without the XML Tags....208
General Author Presentation....209
Author Views....210
The Author Editor....215
Review....230
Profiling / Conditional Text....232

Chapter 6: Author for DITA....237

Creating DITA Maps and Topics....238
Editing DITA Maps....238
Editing Actions....240
Creating a Map....241
Validating DITA Maps....241
Create a Topic in a Map....242
Organize Topics in a Map....242

Create a Bookmap....242
Create a Subject Scheme....243
Create Relationships Between Topics....243
Advanced Operations....244
Transforming DITA Maps and Topics....247
Available Output Formats....247
Configuring a DITA Transformation....249
Running a DITA Map ANT Transformation....250
Customizing a DITA Scenario....250
Set a Font for PDF Output Generated with Apache FOP....256
DITA-OT Customization....256
Support for Transformation Customizations....256
Using Your Custom Build File....256
Customizing the Oxygen Ant Tool....256
Upgrading to a New Version of DITA OT....256
Increasing the Memory for the Ant Process....257
Resolving Topic References Through an XML Catalog....257
DITA Specialization Support....257
Integration of a DITA Specialization....257
Editing DITA Map Specializations....257
Editing DITA Topic Specializations....258
Use a New DITA Open Toolkit in Oxygen....258
Reusing Content....258
Working with Content References....259
How to Work with Reusable Components....259
Insert a Direct Content Reference....260
DITA Profiling / Conditional Text....260
Profiling / Conditional Text Markers....261
Publish Profiled Text....261
Working with MathML....262

Chapter 7: Predefined Document Types....263

Document Type....264
The DocBook 4 Document Type....264
Author Extensions....265
Transformation Scenarios....268
Templates....268
The DocBook 5 Document Type....268
Author Extensions....268
Transformation Scenarios....268
Templates....269
The DocBook Targetset Document Type....269
Templates....269
The DITA Topics Document Type....269

Author Extensions....270
Transformation Scenarios....275
Templates....275
The DITA Map Document Type....276
Author Extensions....276
Transformation Scenarios....276
Templates....277
The XHTML Document Type....277
Author Extensions....277
Transformation Scenarios....278
Templates....279
The TEI P4 Document Type....279
Author Extensions....279
Transformation Scenarios....281
Templates....281
The TEI P5 Document Type....281
Transformation Scenarios....281
Templates....281
The MathML Document Type....282
Templates....282
The Microsoft Office OOXML Document Type....282
Templates....283
The Open Office ODF Document Type....283
Templates....284
The OASIS XML Catalog Document Type....284
Templates....284
The XML Schema Document Type....285
The Relax NG Document Type....285
The NVDL Document Type....285
The Schematron Document Type....285
The Schematron 1.5 Document Type....286
The XSLT Document Type....286
The XMLSpec Document Type....286
Transformation Scenarios....286
Templates....286
The FO Document Type....287
Transformation Scenarios....287
The EAD Document Type....287
Templates....287
The EPUB Document Type....287

Chapter 8: Author Developer Guide....289

Simple Customization Tutorial....291
XML Schema....291

CSS Stylesheet....292
The XML Instance Template....294
Advanced Customization Tutorial - Document Type Associations....295
Author Settings....296
Configuring New File Templates....314
Configuring XML Catalogs....316
Configuring Transformation Scenarios....316
Configuring Validation Scenarios....318
Configuring Extensions....320
Customizing the Default CSS of a Document Type....343
Document Type Sharing....344
CSS Support in Oxygen Author....345
CSS 2.1 Features....345
CSS 3 Features....349
Additional Custom Selectors....353
Oxygen CSS Extensions....354
Example Files Listings - The Simple Documentation Framework Files....360
XML Schema files....360
CSS Files....362
XML Files....364
XSL Files....366
Oxygen XML Author Component....367
Licensing....367
Installation Requirements....368
Customization....368
Deployment....370

Chapter 9: Grid Editor....373

Layouts: Grid and Tree....375
Navigating the Grid....375
Specific Grid Actions....376
Sorting a Table Column....376
Inserting a Row in a Table....376
Inserting a Column in a Table....376
Clearing the Content of a Column....376
Adding Nodes....376
Duplicating Nodes....377
Refresh Layout....377
Start Editing a Cell Value....377
Stop Editing a Cell Value....377
Drag and Drop in the Grid Editor....377
Copy and Paste in the Grid Editor....377
Bidirectional Text Support in the Grid Editor....379

Chapter 10: Transforming Documents.....381

Output Formats.....382
Transformation Scenario.....383
Batch Transformation.....383
Built-in Transformation Scenarios.....383
Defining a New Transformation Scenario.....384
Sharing the Transformation Scenarios.....393
Transformation Scenarios View.....393
XSLT Processors.....394
Supported XSLT Processors.....394
Configuring Custom XSLT Processors.....396
Configuring the XSLT Processor Extensions Paths.....396
XSL-FO Processors.....396
The Built-in XSL-FO Processor.....396
Add a Font to the Built-in FOP - The Simple Version.....397
Add a Font to the Built-in FOP.....398
XProc Transformations.....400
XProc Transformation Scenario.....400
Integration of an External XProc Engine.....401

Chapter 11: Querying Documents.....403

Running XPath Expressions.....404
What is XPath.....404
Oxygen's XPath Console.....404
The XPath Builder View.....407
Working with XQuery.....408
What is XQuery.....408
Syntax Highlight and Content Completion.....408
XQuery Outline View.....409
The XQuery Input View.....410
XQuery Validation.....411
Other XQuery Editing Actions.....412
Transforming XML Documents Using XQuery.....412

Chapter 12: Debugging XSLT Stylesheets and XQuery Documents.....417

Overview.....418
Layout.....418
Control Toolbar.....419
Information View.....421
Multiple Output Documents in XSLT 2.0.....430
Working with the XSLT / XQuery Debugger.....431
Steps in a Typical Debug Process.....431

Using Breakpoints....431
Determining What XSLT / XQuery Expression Generated Particular Output....432
Debugging Java Extensions....434
Supported Processors for XSLT / XQuery Debugging....435

Chapter 13: Profiling XSLT Stylesheets and XQuery Documents....437
Overview....438
Viewing Profiling Information....438
Invocation Tree View....438
Hotspots View....439
Working with XSLT/XQuery Profiler....439

Chapter 14: Comparing and Merging Documents....441
Directories Comparison....442
Directories Comparison User Interface....442
Comparison Result....443
Compare Images....444
Files Comparison....444
Main Menu....445
Compare Toolbar....447
Files Selector....448
File Contents Panel....448
Word Level Comparison....448
Character Level Comparison....449
XML Diff API....449

Chapter 15: Working with Archives....451
Browsing and Modifying Archive Structure....452
Working with EPUB....453
Create an EPUB....454
Publish to EPUB....454
Editing Files From Archives....454

Chapter 16: Working with Databases....455
Relational Database Support....456
Configuring Database Data Sources....456
Configuring Database Connections....461
Resource Management....466
SQL Execution Support....471
Native XML Database (NXD) Support....473
Configuring Database Data Sources....474
Configuring Database Connections....476

Data Source Explorer View....478
XQuery and Databases....488
Build Queries With Drag and Drop From Data Source Explorer View....488
XQuery Transformation....488
XQuery Database Debugging....490
WebDAV Connection....491
How to Configure a WebDAV Connection....491
WebDAV Connection Actions....492

Chapter 17: Importing Data....493
Introduction....494
Import from Database....494
Import Table Content as XML Document....494
Convert Table Structure to XML Schema....496
Import from MS Excel Files....496
Import from HTML Files....497
Import from Text Files....497

Chapter 18: Content Management System (CMS) Integration....501
Integration with Documentum (CMS)....502
Configure Connection to Documentum Server....502
Documentum (CMS) Actions in the Data Source Explorer View....503
Transformations on DITA Content from Documentum (CMS)....507

Chapter 19: Composing Web Service Calls....509
Overview....510
Composing a SOAP Request....510
Testing Remote WSDL Files....513
The UDDI Registry Browser....513
Generate WSDL Documentation....514

Chapter 20: Digital Signatures....517
Overview....518
Canonicalizing Files....519
Certificates....520
Signing Files....520
Verifying the Signature....521

Chapter 21: Syncro SVN Client....523
Main Window....524
Views....524

Main Menu....524
Main Toolbar....529
Status Bar....530
Getting Started....531
SVN Repository Location....531
Defining a Working Copy....532
Manage Working Copy Resources....536
Synchronize with Repository....539
Obtain Information for a Resource....548
Management of SVN Properties....551
Branches and Tags....551
Working with Repositories....574
Sparse Checkout....576
Syncro SVN Client Views....576
Repositories View....576
Working Copy View....579
History View....588
The Editor Panel of SVN Client....591
Annotations View....591
Compare View....592
Image Preview....594
Compare Images View....594
Properties View....594
Console View....596
Dynamic Help View....596
The Revision Graph of a SVN Resource....596
Syncro SVN Client Preferences....599
Command Line Reference....600
Checkout Command....600
Update Command....600
Commit Command....600
Diff Command....600
Show History....601
Refresh....601
Synchronize....601
Import....601
Export....601
Information....602
Add....602
Add to svn:ignore....602
Delete....602
Copy....602
Move / Rename....603
Mark resolved....603
Revert....603

Cleanup....603
Show / Refresh Properties....603
Branch / Tag....604
Merge....604
Scan for locks....604
Lock....605
Unlock....605
Mark as merged....605
Override and update....605
Override and Commit....605
Add / Edit property....605
Remove property....606
Revert changes from this revision....606
Revert changes from these revisions....606

Chapter 22: Extending Oxygen with Plugins....607

Introduction....608
Oxygen Plugins....608
General Plugin....609
Selection Plugin....609
Document Plugin....610
Custom Protocol Plugin....610
Resource Locking Custom Protocol Plugin....610
Components Validation Plugin....611
Workspace Access Plugin....612
Open Redirect Plugin....613
How to Write a CMS Integration Plugin....613
How to Write A Custom Protocol Plugin....617
How to Install a Plugin....617
Example - A Selection Plugin....618

Chapter 23: Text Editor Specific Actions....621

Undoing and Redoing User Actions....622
Copying and Pasting Text....622
Finding and Replacing Text in the Current File....622
The Find / Replace Dialog....622
The Find All Elements / Attributes Dialog....624
The Quick Find Toolbar....625
Keyboard Shortcuts for Finding the Next and Previous Match....626
Finding and Replacing Text in Multiple Files....626
Spell Checking....629
Spell Checking Dictionaries....630
Learned Words....631

Ignored Words....631
Automatic Spell Check....631
Spell Checking in Multiple Files....632
Changing the Font Size....632
Word/Line Editor Actions....633
Dragging and Dropping the Selected Text....633
Inserting a File at Caret Position....633
Opening the File at Caret in System Application....633
Opening the File at Caret Position....633
Switching Between Opened Tabs....633
Printing a File....634
Exiting the Application....635

Chapter 24: Configuring the Application....637

Importing / Exporting Global Options....638
Preferences....638
Global....638
Fonts....640
Document Type Association....641
Perspectives Layout....643
Encoding....644
Editor....645
CSS Validator....675
XML....676
Data Sources....698
SVN....702
Files Comparison....706
Directories Comparison....707
Archive....708
Plugins....710
External Tools....710
Menu Shortcut Keys....712
File Types....713
SVN File Editors....714
Custom Editor Variables....715
HTTP(S) / (S)FTP / Proxy Configuration....716
Certificates....718
XML Structure Outline....719
View....719
Messages....720
Tree Editor....721
Sharing Preferences....721
Automatically Importing the Preferences from the Other Distribution....722
Reset Global Options....722

Scenarios Management....722
Editor Variables....722
Custom Editor Variables....723
Configure Toolbars....724

Chapter 25: Common Problems....725

XML Document Opened After a Long Time....727
Out Of Memory Error When I Open Large Documents....727
Special Characters Are Replaced With a Square in Editor....727
XSLT Debugger Is Very Slow....727
The Scroll Function of my Notebook's Trackpad is Not Working....728
NullPointerException at Startup on Windows XP....728
Crash at Startup on Windows with an Error Message About a File nvoglv32.dll....728
Oxygen Crashed on My Mac OS X Computer....729
Wrong Highlights of Matched Words in a Search in User Manual....729
Keyboard Shortcuts Do Not Work....729
After Installing Oxygen I Cannot Open XML Files in Internet Explorer Anymore....729
I Cannot Associate Oxygen With a File Type on My Windows Computer....730
The Files Are Opened in Split Panels When I Restart the Oxygen Application....730
Grey Window on Linux With the Compiz / Beryl Window Manager....730
Drag and Drop Without Initial Selection Does Not Work....731
Set Specific JVM Version on Mac OS X....731
Segmentation Fault Error on Mac OS X....731
Damaged File Associations on Mac OS X....731
I Cannot Set XML Files to Open With Oxygen....732
I Cannot Connect to SVN Repository From Repositories View....732
Problem Report Submitted on the Technical Support Form....733
Cannot cancel a system shutdown when there is at least one modified document open in Oxygen....733

Chapter 1

Introduction

Topics:

- *Key Features and Benefits of Oxygen XML Editor*

Welcome to the Oxygen XML Editor 12.2 User Manual.

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

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

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

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

Key Features and Benefits of Oxygen XML Editor

Multiplatform availability: Windows, Mac OS X, Linux, Solaris	Multilanguage support: English, German, French, Italian and Japanese
Visual WYSIWYG XML editing mode based on W3C CSS stylesheets.	Visual DITA Map editor
Closely integrate with the DITA Open Toolkit for generating DITA output	Support for latest versions of document frameworks: DocBook and TEI.
Can be used as standalone desktop application, run through Java Web Start or as an Eclipse plugin	Non blocking operations, you can perform validation and transformation operations in background
Support for XML, XML Schema, Relax NG , Schematron, DTD, NVDL schemas, XSLT, XSL:FO, WSDL, XQuery, HTML, CSS	Support for XML, CSS, XSLT, XSL-FO.
Validate XML Schema schemas, Relax NG schemas, DTD's, Schematron schemas, NVDL schemas, WSDL, XQuery, HTML and CSS	Manual and automatic validation of XML documents against XML Schema schemas, Relax NG schemas, DTD's, Schematron, and NVDL schemas
Multiple built-in validation engines (Xerces, libxml, Saxon SA, MSXML 4.0, MSXML.NET) and support for custom validation engines (XSV, SQC).	Multiple built-in XSLT transformers (Saxon 6.5, Saxon 9 Enterprise (schema aware), Xalan, libxslt, MSXML 3.0 / 4.0, Microsoft .NET 1.0, Microsoft .NET 2.0), support for custom JAXP transformers.
Support for latest versions of document frameworks: DocBook and TEI.	Compare and merge files and directories
Ready to use FOP support to generate PDF or PS documents	XInclude support
Support for editing remote files over FTP, SFTP, HTTP / WebDAV and HTTPS / WebDAV	Easy error tracking - locate the error source by clicking on it
Visual schema editor with full and logical model views	Generate HTML documentation from XML Schemas
New XML document wizards to easily create documents specifying a schema or a DTD	Context sensitive content assistant driven by XML Schema, Relax NG, DTD, NVDL or by the edited document structure enhanced with schema annotation presenter
XML Catalog support	Unicode support
Conversions from DTD, Relax NG schema or a set of documents to XML Schema, DTD or Relax NG schema	Syntax coloring for XML, DTD, Relax NG compact syntax, Java, C++, C, PHP, Perl, etc
Pretty-printing of XML files	Easy configuration for external FO Processors
Apply XSLT and FOP transformations	XPath search and evaluation support
Preview transformation results as XHTML or XML or in your browser	Support for document templates to easily create and share documents
Import data from a database, Excel, HTML or text file	Convert database structure to XML Schema
Canonicalize and sign documents	XML project manager
Batch validate selected files in project	Fully-fledged client for the Subversion (SVN) versioning system with support for SVN 1.3, SVN 1.4, SVN 1.5 and SVN 1.6 repositories.

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

Chapter 2

Installation

Topics:

- *Installation Requirements*
- *Installation Instructions*
- *Java Web Start (JWS) Installer*
- *Setting a Parameter in the Launcher Configuration File / Startup Script*
- *Starting the Application*
- *Obtaining and Registering a License Key*
- *Unregistering the License Key*
- *Upgrading the Oxygen Application*
- *Checking for New Versions*
- *Uninstalling the Application*
- *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 the application.

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

Installation Requirements

This section contains details about the platform and environment requirements necessary for installing and running the application.

Platform Requirements

The run-time requirements of the application are:

- 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™ and Windows 7) ; recommended - 2 GB of RAM.
- Hard disk space: minimum - 300 MB free disk space ; recommended - 500 MB free disk space.

Operating System

Windows	Windows XP, Windows Vista, Windows 7, Windows 2003, Windows Server 2008
Mac OS	Mac OS X version 10.4 or later
Unix/Linux	Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.5 or 1.6 from Oracle (formerly from Sun).

Environment Requirements

This section specifies the Java platform requirements and other tools that may be needed for installing the application.

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 on *the Download page* for every archive. You should check the MD5 sum of the downloaded archive with a MD5 checking tool available on your platform.

Java Virtual Machine Prerequisites

Prior to installation ensure that your Operating System environment complies with the following:

- Oxygen XML Editor supports only official and stable Java virtual machines with the version number 1.5.0 or later from Oracle, formerly Sun Microsystems (available at *http://www.java.com/en/download/manual.jsp*) 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 may work very well with JVM implementations from other vendors but the eventual incompatibilities will not be solved in further Oxygen XML Editor releases. Oxygen XML Editor *does not work with the GNU libgcj Java virtual machine*.
- The PATH environment variable is set to the most current Java Virtual Machine installation.
- References to older Java Virtual Machine installations are removed from the PATH.

JWS-specific Requirements

For **Windows** and **Linux** installations:

- The minimum Java Virtual Machine version should be 1.6 update 10 or later;

- The browser should support the New Java™ Plug-In Technology (usually this plug-in is installed together with the Java VM).

For **Mac OS X** installations, note that the New Java™ Plug-In Technology support is not enabled by default on Mac OS X. To enable the plugin follow these steps:

- Upgrade to Java VM version 1.6 update 17 or later;
- Open the **Java Preferences** application and toggle setting to *Run applets: in their own process*;
- Restart your **Safari** browser and try viewing the applet.

Caution:

If you want to run Oxygen with Java Web Start directly from *Oxygen Java Web Start page* or from your intranet server please configure your Java Web Start not to ask for desktop integration (File -> Preferences, Shortcuts). If you don't, the application will freeze because it will show up a dialog in the same time with the Oxygen license registration dialog.

Installation Instructions

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

Note:

The following instructions assume that a Java Runtime Environment *JRE*) is installed. If you have downloaded an installation package that contains the *JRE*, please note that the package will automatically install a JRE before execution of the application but this JRE will be used on your computer only for running Oxygen , it will be invisible to other applications.

Note:

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.

Windows Installation

Windows installation procedure.

To install the application on Windows:

1. Download the `oxygen.exe` installation kit and run it.
2. Follow the instructions presented in the installation program.

   The user preferences are stored in the subfolder `com.oxygenxml` of the folder that is the value of the APPDATA Windows variable for the user that starts the application.

   Note:

   In order to specify another Java virtual machine to be used by Oxygen you have to set the home folder of the desired JVM in the Windows variable JAVA_HOME or in the Windows variable JDK_HOME. If JAVA_HOME and JDK_HOME are not set the application launcher will try to detect a JVM installed in a standard location on the computer and use it for running the application. If you installed the kit which includes a Java virtual machine you have to rename of remove the `jre` subfolder of the install folder in order for the variable JAVA_HOME or JDK_HOME to have an effect.

   Note:

   Copy to clipboard the license key you have received by email and paste it in the application license dialog.

Note:

If you have difficulties installing the product please use the zip archive distribution instead (oxygen-32bit.zip). Just unzip it in a folder where you have write permissions and use the product launchers.

Mac OS X Installation

Mac OS X installation procedure.

To install the application on Mac OS X:

1. Create a folder called `oxygen` on your local disk.
2. Within the `oxygen` folder, create child folder named in accordance with the version number of the application.

   The folder structure looks as follows: /../oxygen/12.2/
3. Download the Mac OS X Installation package (`oxygen.tar.gz`) into this folder.
4. Extract the archive into the same folder.
5. Execute the file named `oxygen`

Note:

Oxygen uses the first JVM from the list of preferred JVM versions set on your Mac computer that has the version number not less than 1.5.0. To change the version of the Java virtual machine that runs the application you must move your desired JVM version up in the preferred list by dragging it with the mouse on the first position in the list of JVMs available from Applications -> Utilities -> Java -> Java Preferences.

Note:

Runs perfectly on Intel Core Duo Mac systems.

Note:

The compressed file should be recognized by StuffIt Expander and should automatically be expanded after downloading. If it is not expanded, you can expand it manually using StuffIt Expander 6.0 or later.

Note:

Copy to clipboard the license key you have received by email and paste it in the application license dialog.

Note:

With some fonts the cursor will behave unpredictably. It is recommended to use a fixed-size font, such as Monaco. More details can be found *here*.

Linux Installation

Linux installation procedure.

To install the application on Linux:

1. Download the `oxygen.sh` installation kit and run it.
2. Follow the instructions presented in the installation program.

Note:

In order to specify another Java virtual machine to be used by Oxygen you have to set the home folder of the desired JVM in the environment variable JAVA_HOME or in the environment variable JDK_HOME. If JAVA_HOME and JDK_HOME are not set the application launcher will try to detect a JVM installed in a standard location on the computer and use it to run the application.

All Platforms Installation

All Platforms kit installation procedure.

1. Create a folder called `oxygen` on your local disk.
2. Within the `oxygen` folder, create child folder named in accordance with the application version number. The directory structure looks as follows: /../oxygen/12.2/
3. Download the All Platforms Installation package (`oxygen.tar.gz`) to this folder.
4. Extract the archive to the same folder.
5. Run from a command line the script `oxygen.bat` on Windows, `oxygenMac.sh` on Mac OS X, `oxygen.sh` on Unix / Linux.

   👉 **Note:**

   To change the version of the Java virtual machine that runs the application you have to specify the full path to the Java executable of the desired JVM version in the Java command at the end of the script file, for example:

   ```
   "C:\Program Files\Java\jre1.5.0_13\bin\java" -Xmx256m
   -Dsun.java2d.noddraw=true ...
   ```

   on Windows,

   ```
   /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java
    "-Xdock:name=Oxygen" ...
   ```

   on Mac OS X.

   👉 **Note:**

   You must have installed a Java virtual machine 1.5 or later.

   👉 **Note:**

   Copy to clipboard the license key you have received by email and paste it in the application license dialog.

Unix / Linux Server Configuration

Unix / Linux Server configuration procedure.

To install the application on a Unix / Linux server:

1. Install the application on the server, making sure the `oxygen.sh` script is executable and the installation directory is in the PATH of the users that need to use the editor.
2. Create a file called `oxygen.vmoptions` in the Oxygen install folder where the oxygen12.2 file is located.

   File's content must be: `-Xmx256m -Dcom.oxygenxml.MultipleInstances=true`

   The "-Xmx" value represents the maximum memory allocated for each application instance. Please make sure you tune it such that the multiple application instances will not use all the available physical memory.
3. Make sure the X server processes located on the workstations allow connections from the server host. For this use the **xhost** command.
4. Telnet (or ssh) on the server host.
5. Start an xterm process, with **display** parameter set on the current workstation. Ex: `xterm -display workstationip:0.0`
6. Start the application by typing `oxygen.sh`

Windows NT Terminal Server Configuration

Windows NT Terminal Server configuration procedure.

1. Install the application on the server, making its shortcuts available to all users.
2. Edit the `oxygen.vmoptions` file located in the install folder, adding the parameter *-Dcom.oxygenxml.MultipleInstances=true* so that the file content looks like: `-Xmx256m -Dcom.oxygenxml.MultipleInstances=true`

   The "-Xmx" value represents the maximum memory allocated for each application instance. Please make sure you tune them in a way that the multiple application instances will not use all the available physical memory.

Unattended Installation

Unattended installation is possible only on Windows and Linux by running the installer executable from command line and passing the -q parameter. The installer executable is called `oxygen.exe` on Windows and `oxygen.sh` on Linux

In unattended mode the installer does not overwrite files with the same name if a previous version of the application is installed in the same folder. The -overwrite parameter added after -q parameter forces the overwriting of these files.

If the installer is executed in silent (unattended) mode and -console parameter is passed as a second parameter after -q parameter, a console will be allocated on Windows that displays the output of the installer. The command for running the installer in this case is:

```
start /wait oxygen.exe -q -console
```

Custom Settings in Unattended Installation

By default an unattended installation applies the default settings of the installer. If you want to install the application on a large number of computers but you need to change the default values of some settings (like the install folder on disk, whether a desktop icon or a quick launch shortcut is created, the file associations created in the operating system, the name of the program group on the Start menu, etc.) then you should use a special settings file which specifies the new values for these settings. To generate the settings file you have to run the installer in normal attended mode once on a test computer and specify the exact options that you want for the unattended installation. When the installation is completed a file called `response.varfile` and containing your selected options is created in the `.install4j` subfolder of the installation folder, by default C:\Program Files\Oxygen XML Editor \.install4j on Windows. This is a one time process. After that for applying these options on all the computers where an unattended installation is performed you have to specify this file in the command line, for example copy the file in the same location as the installer program and use the command:

- on Windows:

```
oxygen.exe -q -varfile response.varfile
```

- on Linux:

```
oxygen.sh -q -varfile response.varfile
```

Privacy Options

The following parameters control the privacy options:

Table 1: Privacy Options Parameters

Parameter name	Description	Values
autoVersionChecking	Automatic version checking.	`true` / `false`. Default setting is `true`.
reportProblem	Allows you to report a problem encountered while using the application.	`true` / `false`. Default setting is `true`.

Parameter name	Description	Values
usageDataCollector	When set to `true`, the user can choose whether the application will send anonymous information about how the product is used.	`true` / `false`. Default setting is `true`.

They can be set either in the `response.varfile` configuration file, or directly in the command line, like in the following example:

- on Windows:

```
oxygen.exe -q -VautoVersionChecking=false
-VreportProblem=false -VusageDataCollector=false
```

- on Linux:

```
oxygen.sh -q -VautoVersionChecking=false
-VreportProblem=false -VusageDataCollector=false
```

Java Web Start (JWS) Installer

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

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

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

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

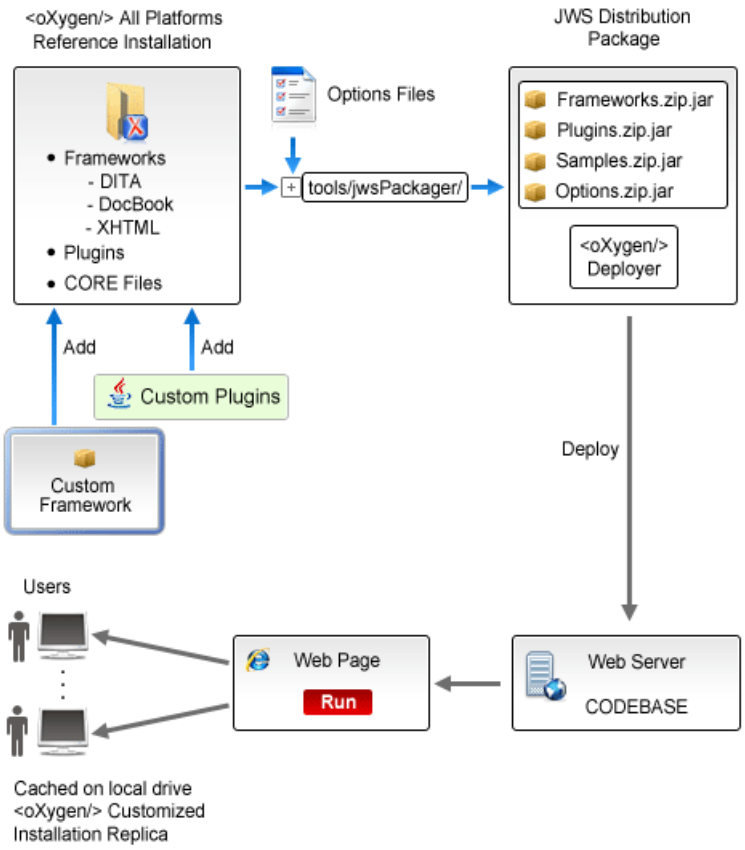


Figure 1: Java Web Start Deployment Procedure

The following steps describe the procedure of deploying a Oxygen installation on a custom server.

1. Download the All Platforms Installation package from `http://www.oxygenxml.com/InstData/Editor/All/oxygen.tar.gz` to a local drive.
2. Expand the archive.
   `oxygen` folder is created.
3. Optionally, you can customize the content of the **frameworks** folder, containing default template document files.
4. Edit the `oxygen\tools\jwsPackager\packager.properties` configuration file. The following properties need to be adjusted:

   - **codebase** - represents the location of the future JWS distribution;
   - **keystore** - keystore location path;
   - **storepass** - password for keystore integrity;
   - **alias** - keystore alias;
   - **optionsDir** - points to the options directory that may be distributed with the JWS installer.

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

   The values of **keystore**, **storepass** and **alias** properties are all provided by the code signing certificate. For more information, please check the documentation of the *`jarsigner`* tool.
5. Edit the JNLP `oxygen\tools\jwsPackager\dist\javawebstart\oxygen\oxygen.jnlp` template file to modify default settings. You can specify the list of files the application opens at startup by modifying the `<argument>` list. To pass system properties directly to the started Oxygen application, you must add them the `oxy`

prefix, like in the example:`<property name="oxyPropertyName" value="testValue"/>`. The system property is passed to the Oxygen application with the prefix stripped.

6. Using a command line console, run `ant` in the `oxygen\tools\jwsPackager` folder.
   The **ant** process creates the `oxygen\tools\jwsPackager\dist\InstData\oxygenJWS.zip` archive that contains the actual remote JWS installer.
7. Copy the expanded content of the archive into the folder specified in the **codebase** property, previously set in the **`packager.properties`** file.
8. Using your favourite web browser, go to the address specified in the **codebase** property or to its parent folder and start the remote installer.

Setting a Parameter in the Launcher Configuration File / Startup Script

On the Windows platform if you start the application by double-clicking on the Start menu shortcut/Desktop shortcut in order to set a startup parameter you have to add a new line with the parameter to the file `oxygen.vmoptions` located in the installation directory together with the launcher file called `oxygen.exe`. If the file `oxygen.vmoptions` does not exist yet in the folder of the launcher file you have to create it there. For example for setting the maximum amount of Java memory to 600 MB the content of the file `oxygen.vmoptions` must be:

```
-Xmx600m
```

Note: On Windows Vista/7 you will first have to copy the `oxygen.vmoptions` file to a folder with write access (like your Desktop), modify it there with a text editing application (like *Notepad*) and then copy it back to the installation folder, replacing the original file.

If you start the application with the script `oxygen.bat` you have to add or modify the parameter to the java command at the end of the script. For example for setting the maximum amount of Java memory to 600 MB the java command should start with:

```
java -Xmx600m -Dsun.java2d.noddraw=true ...
```

On the Mac OS X platform to add or modify a startup parameter you have to Ctrl-click on the Oxygen application icon in Finder, in the pop-up menu select *Show Package Contents*, then in the *Contents* directory you edit the file `Info.plist`: in the key *VMOptions* you modify the parameter if it already exists in that key or you add it after the model of the existing parameters inside that key.

On the Linux platform you have to create a file called `oxygen.vmoptions` if it does not exist already and specify the parameter exactly as in the case of the `.vmoptions` file on the Windows platform.

If you use the *All platforms* distribution you have to add or modify the startup parameter that you want to set in the Java command line at the end of the startup script `oxygen.bat` on Windows, `oxygenMac.sh` on Mac OS X and `oxygen.sh` on Linux. All these files are located in the installation directory. For example to set the maximum amount of Java memory to 600 MB on Windows the -Xmx parameter must be modified in the java command line at the end of `oxygen.bat` like this:

```
java -Xmx600m -Dsun.java2d.noddraw=true ...
```

on Mac OS X the java command at the end of `oxygenMac.sh` should look like:

```
java "-Xdock:name=Oxygen"\
 -Dcom.oxygenxml.editor.plugins.dir="$OXYGEN_HOME/plugins"\
 -Xmx600m\
 ...
```

and on Linux the Java command at the end of `oxygen.sh` should look like:

```
java -Xmx600m\
 "-Dcom.oxygenxml.editor.plugins.dir=$OXYGEN_HOME/plugins"\
```

Starting the Application

This section specifies the steps for starting the application.

Starting the Application on Windows

Start the application launcher.

Use one of the following two launchers:

- `oxygen.exe` - started from the shortcut created by the installer on the **Start** menu.
- `oxygen.bat` - located in the install folder and started from command line.

Starting the Application on Mac OS X

Start the application's launcher.

Use one of the following two methods:

- The shortcut `Oxygen` created on **Desktop** by the installer.
- The command

  ```
  sh oxygenMac.sh
  ```

  executed from command line. This launcher file is located in the install folder.

: Two or more instances can be started on the same computer with the following command that should be executed for any new instance:

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

Starting the Application on Linux

Start the application's launcher.

Use one of the following two methods:

- The shortcut `oxygen` created on **Desktop** by the installer.
- The command

  ```
  sh oxygen.sh
  ```

  executed from command line. This launcher file is located in the install folder.

Starting the Application with the All Platforms Kit

Start the application's launcher.

Use the following command:

- On Windows:

  ```
  oxygen.bat
  ```

- On Linux:

```
sh oxygen.sh
```

- On Mac OS X:

```
sh oxygenMac.sh
```

Obtaining and Registering a License Key

The Oxygen XML Editor is not free software and requires a license in order to enable the application.

For demonstration and evaluation purposes a time limited license is available upon request from the *Oxygen* web site. This license is supplied at no cost for a period of 30 days from date of issue. During this period the Oxygen XML Editor is fully functional enabling you to test all aspects of the application. Thereafter, the application is disabled and a permanent license must be purchased in order to use the application. For special circumstances, if a trial period of greater than 30 days is required, please contact support@oxygenxml.com . All licenses are obtained from the *Oxygen web site* .

For definitions and legal details of the license types available for Oxygen XML Editor you should consult the End User License Agreement received with the license key and available also on the Oxygen website at *http://www.oxygenxml.com/eula.html*

Named User License Registration

1. Save a backup copy of the message containing the new license key.
2. Start the application.
3. Copy to the clipboard the license text as explained in the message.
4. If this is a new install of the application then it will display automatically the registration dialog when it is started. In the case you already used the application and obtained a new license, use the menu option Help / Register

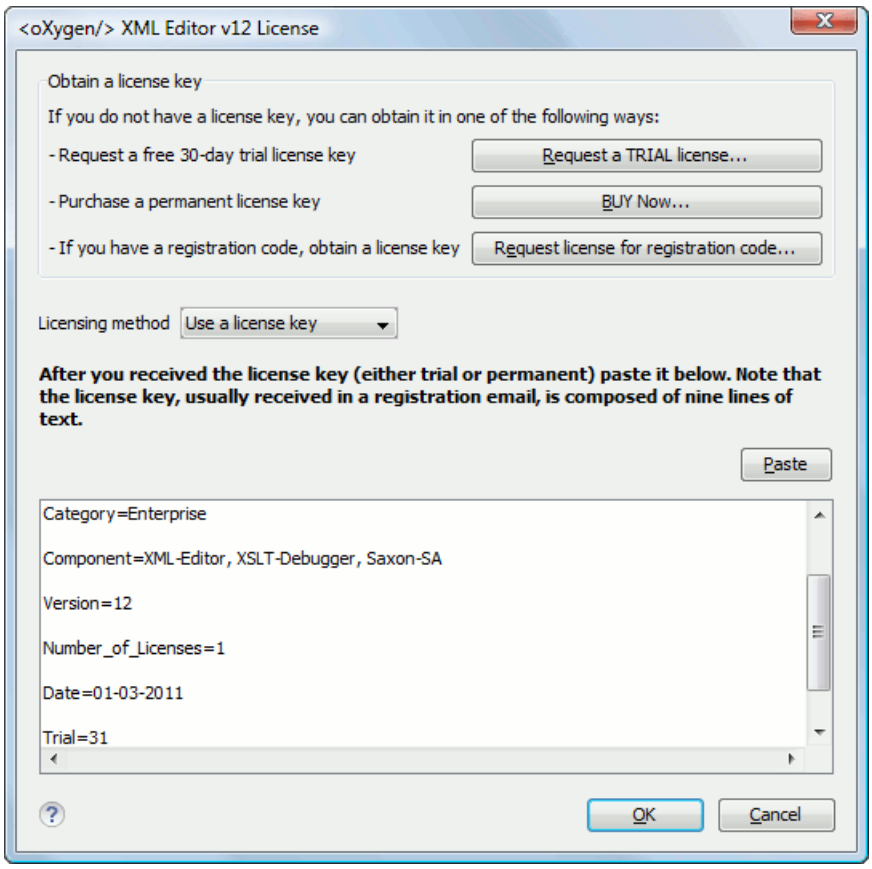


Figure 2: Registration Dialog

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 the file in the *lib* subfolder of the install folder.
3. Start Oxygen .

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 will 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.

The license management is done either by the application itself or by the Oxygen license server:

- if you plan to use the application on machines running in the same local network, Oxygen can manage the licenses usage by itself. Different running instances of the application communicate between them. The registration procedure requires you to paste the license key in the license registration dialog. See *Named User License Registration* procedure for more details.
- if you plan to use the application on machines running in different network segments, then you must use a Oxygen 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. You can get it from: `http://tomcat.apache.org/`

1. Download the license servlet **Web ARchive** (**.war**) 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 then press 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. Restart *oXygen License Servlet* from the Tomcat Web Application Manager page.

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

- you have multiple license keys for the same Oxygen 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 runs
- MAC address of the computer where Oxygen 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.

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:

   - 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 port number used to communicate with Oxygen 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 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.

   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` - install the server as Windows service with the name "oXygenLicenseServer". The parameters for the license key folder and the server port can be set in the `oXygenLicenseServer.vmoptions` file.
- `startWindowsService.bat` - start the Windows service.
- `stopWindowsService.bat` - stop the Windows service.
- `uninstallWindowsService.bat` - uninstall the Windows service created by the `installWindowsService.bat` script.

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.

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

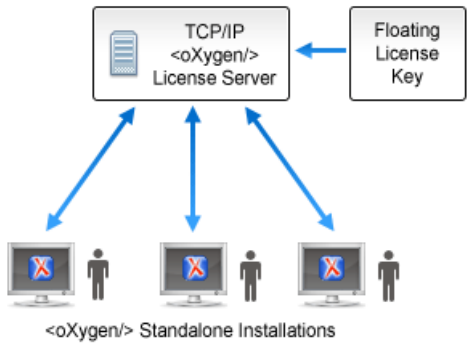


Figure 3: Floating License Server Running as a Standalone Process

1. Start the application.
2. Go to menu **Help** > **Register** .
   The license dialog is displayed.
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 . 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.

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

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

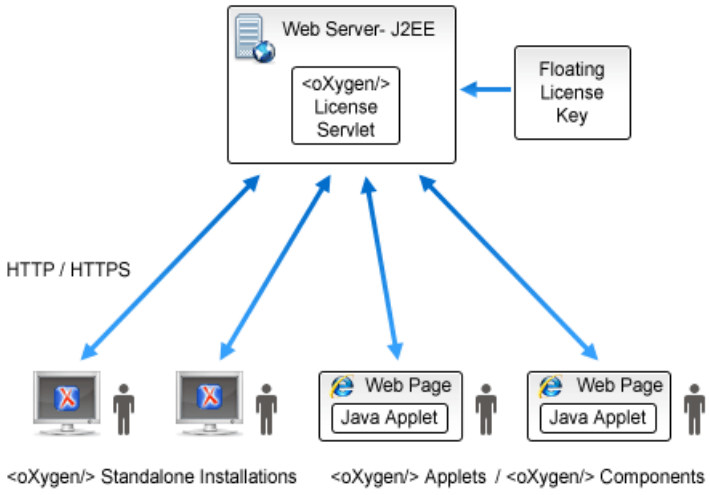


Figure 4: Floating License Server Running as a Servlet

1. Start the application.
2. Go to menu **Help** > **Register** .
   The license dialog is displayed.
3. Choose **Use a license server** as licensing method.
4. Select **HTTP 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 . 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 instances (for example one standalone and one Eclipse plugin) run on the same machine, consume a single license key.

Release a Floating License

The floating license key registered for the current Oxygen instance will be released automatically when the connection with the license server is lost (after closing the application or as a result of a network failure).

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

1. Go to the menu **Help** .
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 website. The button **Request license for registration code** in the registration dialog available from menu **Help** > **Register** > **Register** opens this request form in the default Web browser on your computer.

Unregistering the License Key

Sometimes you 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 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 **Help** > **Register** > **Register**
   This displays the license registration dialog.
2. Make sure the text area for the license key is empty.
3. Make sure the checkbox **Use a license server** is unchecked.
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.

Upgrading the Oxygen Application

From time to time, upgrade and patch versions of Oxygen 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 Standalone Application

Upgrading to a new version might require a new license key. To check if your license key is compatible with the new version, select **Help** > **Check for New Version** . Note that the application needs an Internet connection to check the license compatibility.

👉 **Note:** Any customization that was applied to a *predefined document type* will be removed by an upgrade.

👉 **Note:** If there is a previous version of Oxygen already installed on your computer, it can coexist with the new one, which means you don't have to uninstall it. However, if you choose to uninstall the older version please note that all the files from its install directory will be removed, including any modification in Author frameworks files, XSLT stylesheets, XML catalogs, and templates. If you have modified any files in that directory, it is recommended to keep the older version until you are sure you don't need it anymore. The user preferences are stored in a different directory. They will not be removed and will be imported automatically in the new version at the first application launch.

1. Download the install kit of the new version.
2. Create a new folder under `/../oxygen` e. g. /../oxygen/12.2

3. Extract the content of the install kit into the new folder.
4. If you have defined Oxygen in the system PATH, modify it to point to the new installation folder.
5. Start Oxygen
   This will test that the application can start and that your license is recognized by the upgrade installation.

Checking for New Versions

Oxygen XML Editor offers *the option of checking for new versions* at the *http://www.oxygenxml.com* site when the application is started. If this option is enabled a message dialog will notify the user when new versions are released.

You can check for new versions manually at any time by going to menu **Help** > **Check for New Versions**

Uninstalling the Application

This section contains uninstallation procedures.

Uninstalling the Standalone Application

Caution: The following procedure will remove Oxygen XML Editor from your system. **Please ensure that all valuable data stored in the install folder is saved to another location prior to performing this procedure.**

1. Backup all valuable data from the Oxygen XML Editor installation folder.
2. Remove the application.
   - On Windows use the appropriate uninstaller shortcut provided with your OS.
   - On Mac OS X and Unix manually delete the installation folder and all its contents.
3. 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 folder on Linux / the subfolder `Library/Preferences/com.oxygenxml` of the user home folder on Mac OS X.

Unattended Uninstall

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

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

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

Performance Problems

This section contains the solutions for some common problems that may appear when running the application.

Large Documents

When started from the icon created on the Start menu or the Desktop on Windows or from the shortcut created on the Linux desktop the maximum memory available to Oxygen is set by default to 40% of the amount of physical RAM but not more than 700 MB. When started from the command line scripts the maximum memory is 256 MB. If large documents are edited in Oxygen and you see that performance slows down considerably after some time then a possible cause is that the application needs more memory in order to run properly. You can increase the maximum amount of memory available to Oxygen by *setting the -Xmx parameter in a configuration file* specific to the platform that runs the application.

Attention:

The maximum amount of memory should not be equal to the physical amount of memory available on the machine because in that case the operating system and other applications will have no memory available.

When installed on a multi-user environment such as Windows Terminal Server or Unix/Linux, to each instance of Oxygen will be allocated the amount stipulated in the memory value. To avoid degrading the general performance of the host system, please ensure that the amount of memory available is optimally apportioned for each of the expected instances.

External Processes

The amount of memory allocated to generate PDF output with the built-in Apache FOP processor is controlled by the FOP memory setting available in Oxygen Preferences: *Memory available to the built-in FOP*. In the application throws an *Out Of Memory* error (**OutOfMemoryError**), this is the setting that must be modified to allow more memory for the built-in FOP.

For external XSL-FO processors *configured in* ***Options*** > ***Preferences*** > ***XML*** > ***XSLT/FO/XQuery*** > ***FO Processors*** and for external XSLT processors *configured in* ***Options*** > ***Preferences*** > ***XML*** > ***XSLT/FO/XQuery*** > ***Custom Engines*** and for external tools *configured in* ***Options*** > ***Preferences*** > ***External Tools*** the maximum memory must be set in the command line of the tool with a parameter -Xmx set to the Java virtual machine.

Display Problems on Linux/Solaris

Display problems like screen freeze or momentary menu pop-ups during mouse movements over screen on Linux or Solaris can be solved by *adding the startup parameter* -Dsun.java2d.pmoffscreen=false.

Chapter 3

Getting Started

Topics:

- *Getting Help*
- *Supported Types of Documents*
- *Perspectives*
- *Dockable Views and Editors*

This section will get you started with the editing perspectives of the application.

Getting Help

Online help is available at any time while working in Oxygen by accessing the **Help** > **Help ...** menu which opens the **Help** dialog.

Context sensitive help is available from any dialog or view by pressing the F1 key. This opens the same **Help** dialog directly on a relevant page for the current view or dialog which has the editing focus.

The same help content is available in the view **Window** > **Show View** > **Dynamic Help** (also available from menu **Help** > **Show Dynamic Help View**) which loads automatically the relevant help section for the focused editor, view or dialog.

The name and version of the third-party libraries and frameworks used by Oxygen are listed in the **About** dialog box: **Help** > **About ...** Also you can see here the values of system properties like the version of the Java virtual machine, the location of the user home directory, the Java classpath, etc.

Supported Types of Documents

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

- XML documents and applications
- XSL stylesheets - transformations and debugging support
- Schema languages: XML Schema, 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 interface uses standard interface conventions and components to provide a familiar and intuitive editing environment across all operating systems.

In Oxygen you can work with documents in one of the 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.
Tree Editor perspective	An XML document is viewed and edited as a tree of XML elements.

Editor Perspective

The Editor perspective is used for editing the content of your documents.

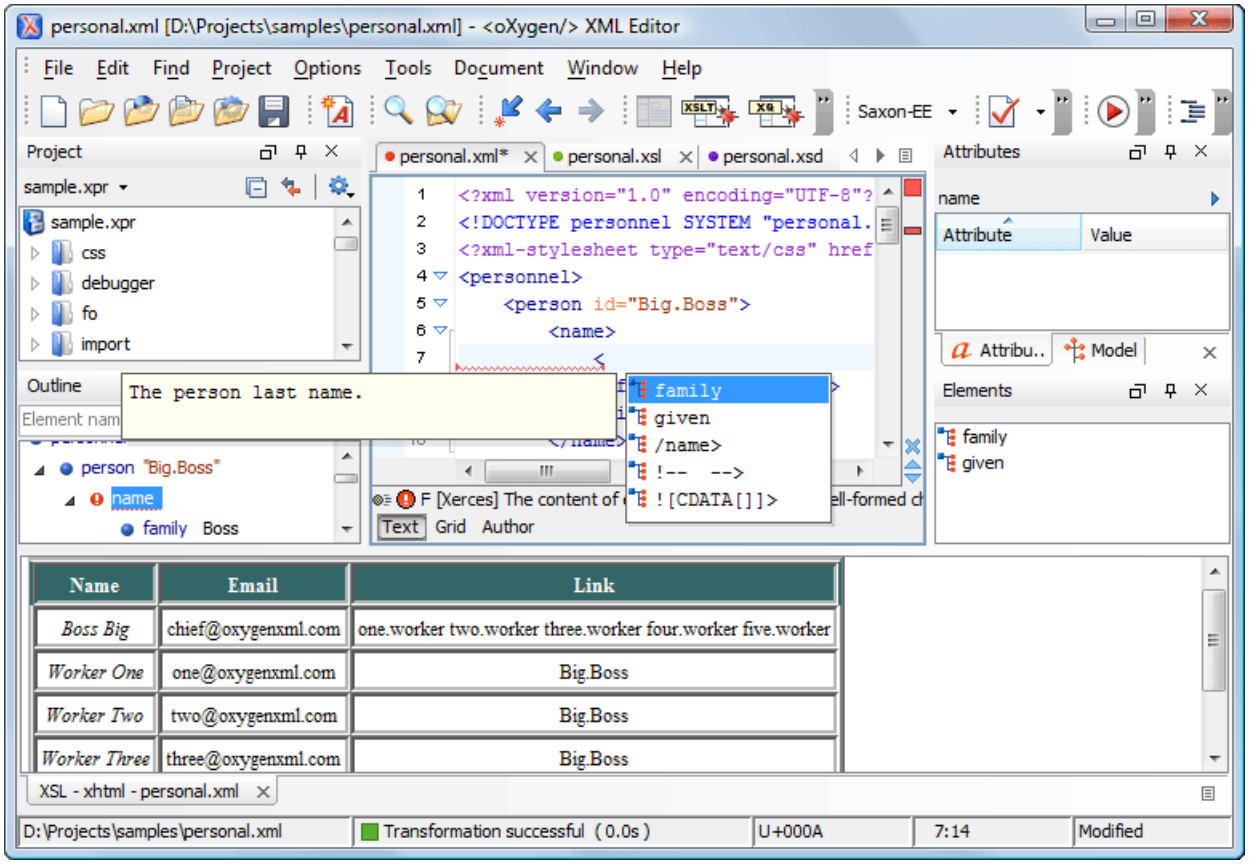


Figure 5: Editor perspective

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

As majority of the work process centers around the Editor panel, other panels can be hidden from view using the expand and collapse controls located on the divider bars.

This perspective organizes the workspace in the following panels:

- **Main menu** - Provides menu driven access to all the features and functions available within Oxygen.
- **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 panel** - The place where you spend most of your time, reading, editing, applying markup and validating your documents.
- **Outline view** - Provides the following functions: XML document overview, modification follow-up, document structure change, document tag selection.
- **Model view panel** - Presents the structure of the current edited tag and additional tag documentation.
- **Results panel** - Displays result messages returned from user operations. The following actions are available:
  - **Hierarchical view** - that allows you to see the results in tree-like manner. Clicking on a tree leaf highlights the corresponding line in the document.
  - **Flat view** - that will present the errors in a table-like manner. Clicking on a table row highlights the corresponding line in the document.
  - **Remove selected** - removes the currently selected message from the list.

- **Remove all** - clears the message list.

  Navigation to the previous and next message is possible from the contextual menu or by using the assigned shortcut keys. The default shortcut keys are Ctrl + Shift +] for navigating to the next and Ctrl + Shift + [for navigating to the previous message.

- **Project view** - Enables the definition of projects and logical management of the documents it contains.

XSLT Debugger Perspective

The XSLT Debugger perspective is used for detecting problems in an XSLT transformation process by executing the process step by step in a controlled environment and inspecting the information provided in different special views. The workspace is organized as an editing area supported by special helper views. The editing area contains editor panels and *can be split horizontally or vertically* in two stacks of editors: XML editor panels and XSLT editor panels.

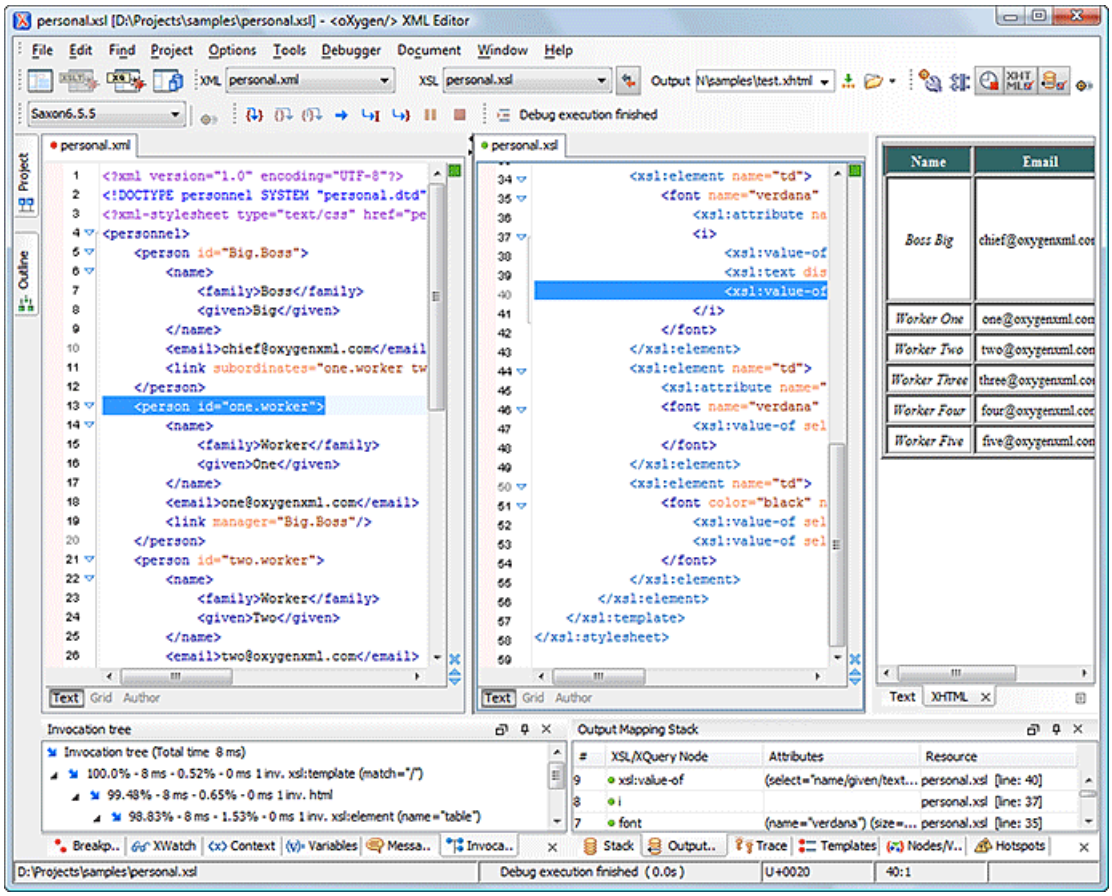


Figure 6: XSLT Debugger perspective

- 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 stylesheet).
- Control toolbar - Contains all actions needed in order to configure and control the debug process.
- Information views - Distributed in two panes that are used to display 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.

XQuery Debugger Perspective

The XQuery Debugger perspective is similar to *the XSLT Debugger perspective.* It is used 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 different special views. The workspace is organized as follows:

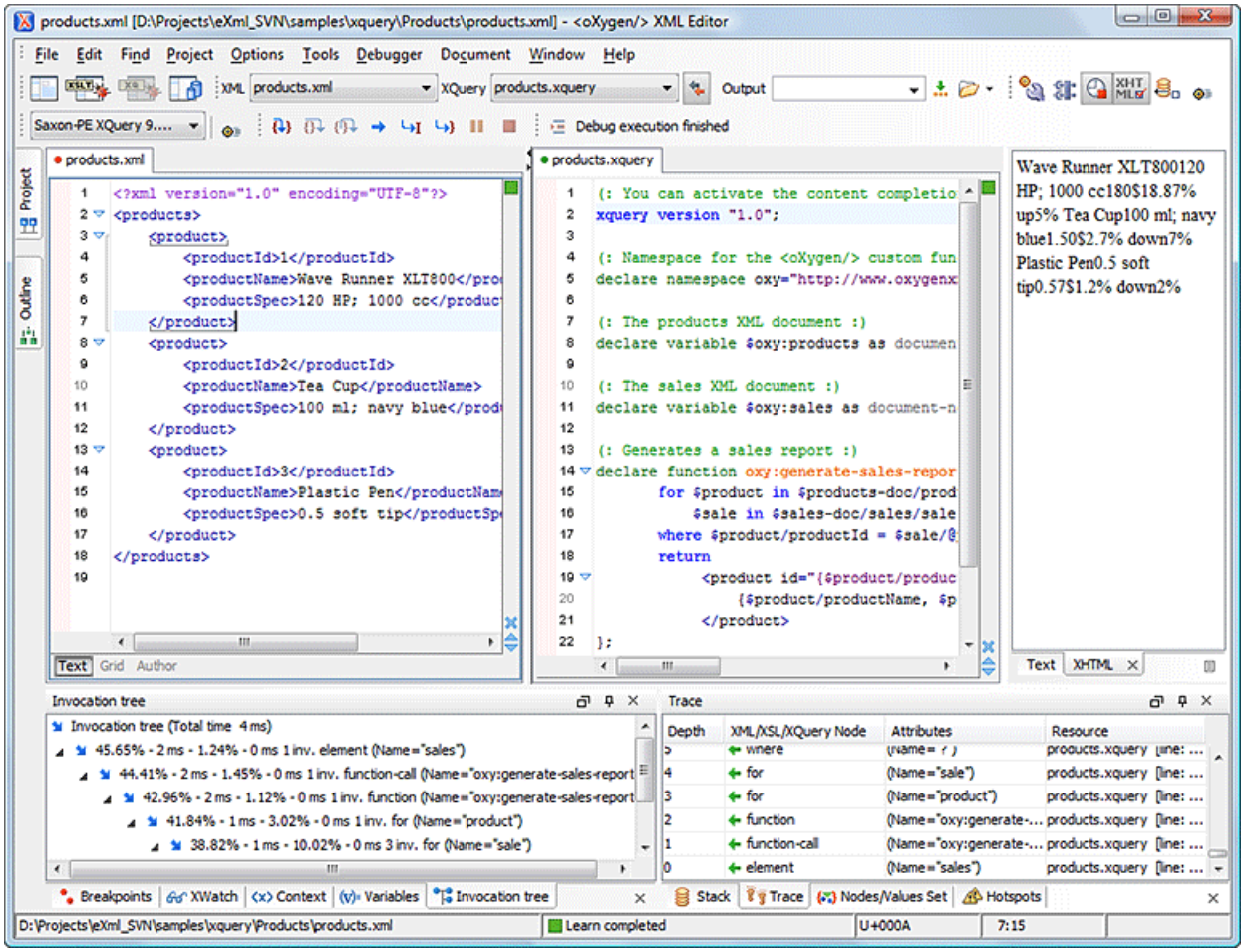


Figure 7: XQuery Debugger perspective

- 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 needed in order to configure and control the debug process.
- Information views - Distributed in two panes that are used to display 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.

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, both 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 (Enterprise edition only)
- MarkLogic (Enterprise edition only, XQuery support 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)
- Software AG Tamino (Enterprise edition only)
- TigerLogic (Enterprise edition only, XQuery support 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. The non-XML capabilities of any database listed here are available also in the Academic and Professional editions of Oxygen 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.

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 editor panel;
- handling the values from columns of type XML Type as XML instance documents that can be opened and edited in an Oxygen 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 please *go to the Oxygen website*.

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

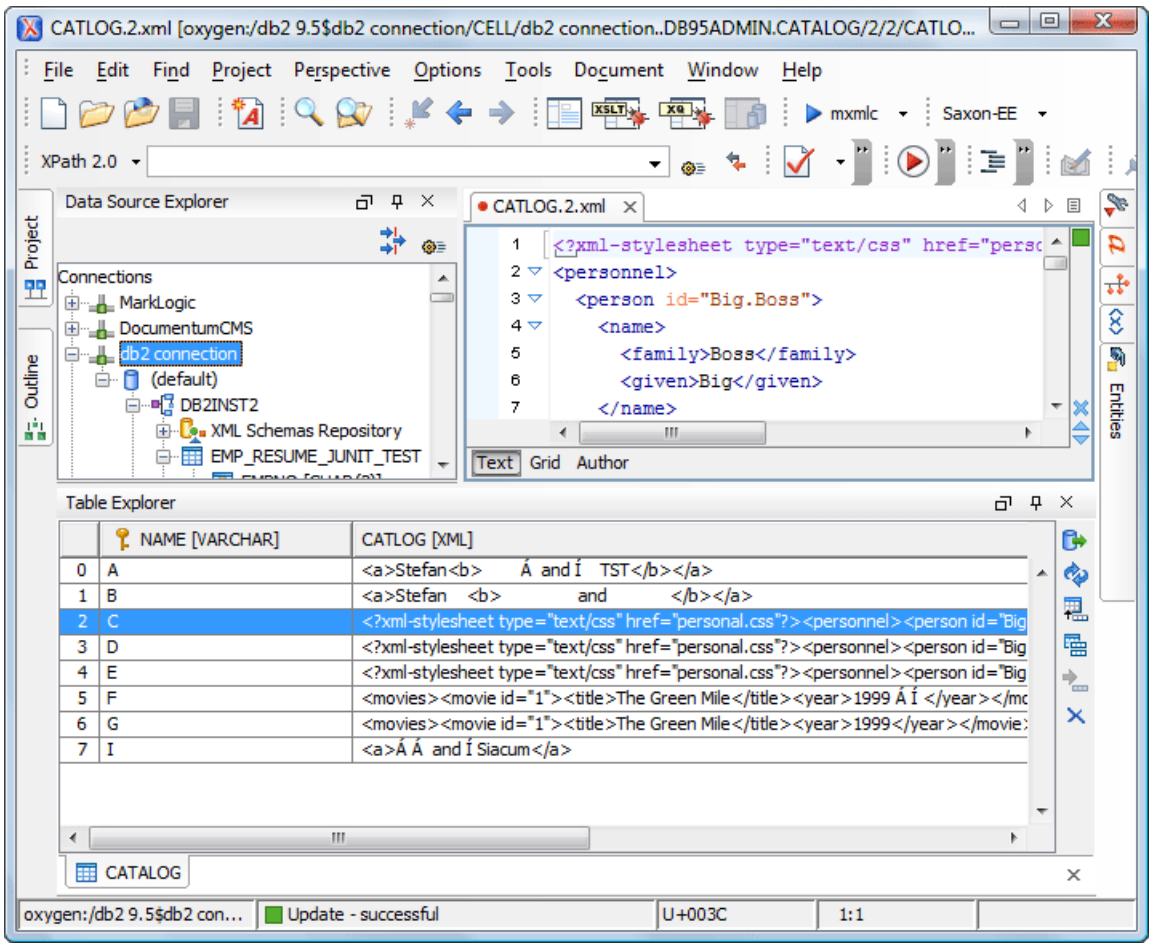


Figure 8: Database perspective

- Main menu - provides menu driven access to all the features and functions available within Oxygen.
- 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 panel - 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: insert a new row, delete a table row, cell value editing, export to XML file.

Tree Editor Perspective

The Tree Editor perspective is used for editing the content of a document viewed as a XML tree. The workspace is organized in:

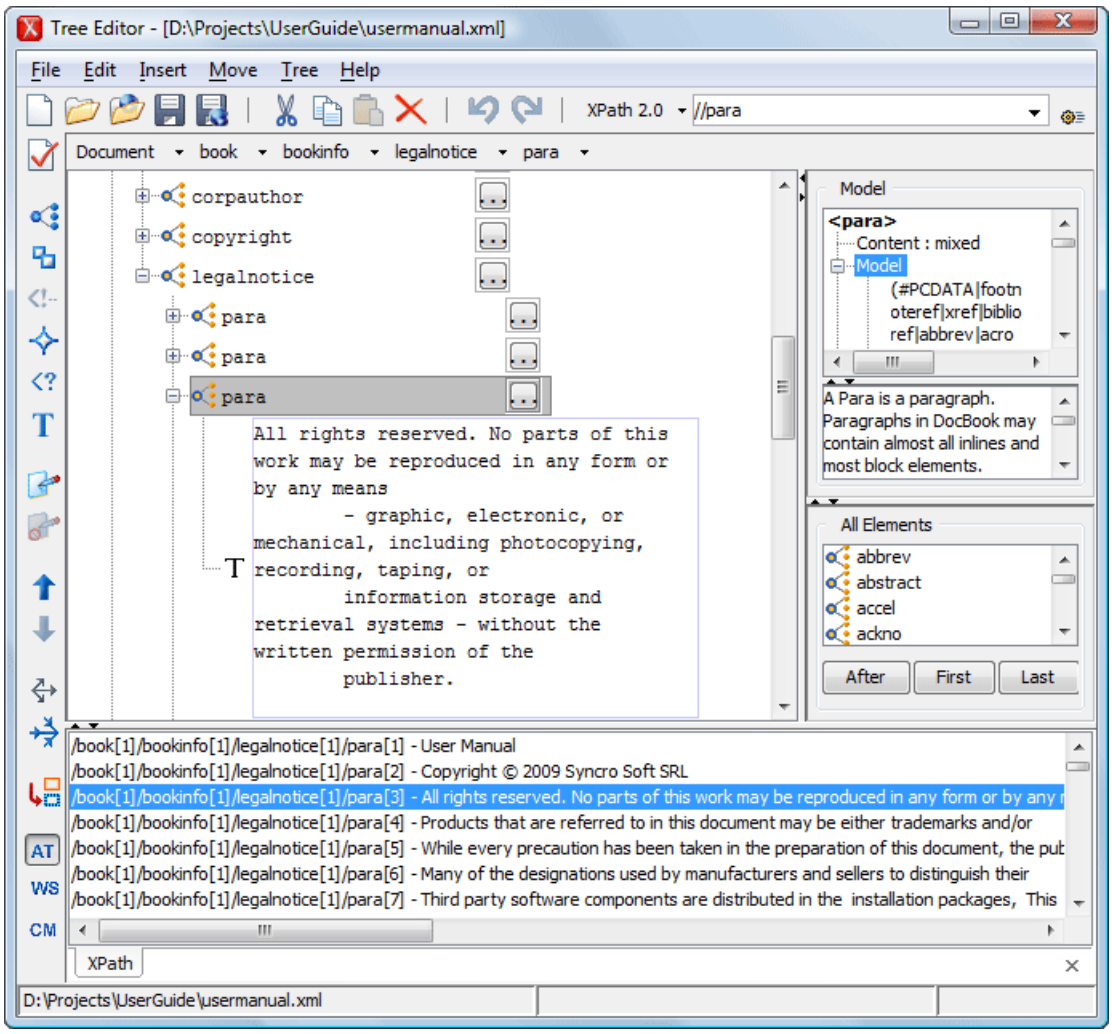


Figure 9: Tree Editor perspective

- Main menu - provides menu driven access to all the features and functions available in Oxygen Tree Editor perspective.
- Toolbar - provides easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function.
- Editor panel - easy editing of structured mark-up documents. Each token has associated an icon for a easy visual identification of the tokens.
- Message panel - displays messages returned from user operations.
- Model panel - shows the detailed information about the attribute or element that you are working on.
- All Elements panel - presents a list of all defined elements that you can insert within your document.

The tree editor does not offer entity support: entities are not presented with a special type of node in the tree and new entity nodes cannot be inserted in the document.

Dockable Views and Editors

All the Oxygen views available in the *Editor Perspective*, *XSLT Debugger Perspective* and *XQuery Debugger Perspective* are dockable. You can drag them to any margin of another view or editor inside the Oxygen window to form any desired layout. Also a view can be set to a floating state to enable it to hover over other views and editors.

To gain more editing space in the Oxygen window, set one or more views to the *auto hide* state: only the title always remains visible, attached to one of the margins of the Oxygen window, while the rest of the view gets restored only when the mouse pointer hovers over the title or when you click the title. The view becomes hidden again when the mouse pointer goes out of the screen area covered by that view.

The editing area can be divided vertically in several editing panels by dragging the title of an editor inside the editing area and dropping it when the frame of the dragged editor is painted in the desired position. In the following figure, you can see how to unsplit the editing area by dragging the title of the `personal.xml` editor panel over personal-schema.xml until the drop frame painted in dark gray covers all the `personal-schema.xml` editor panel and then dropping it.

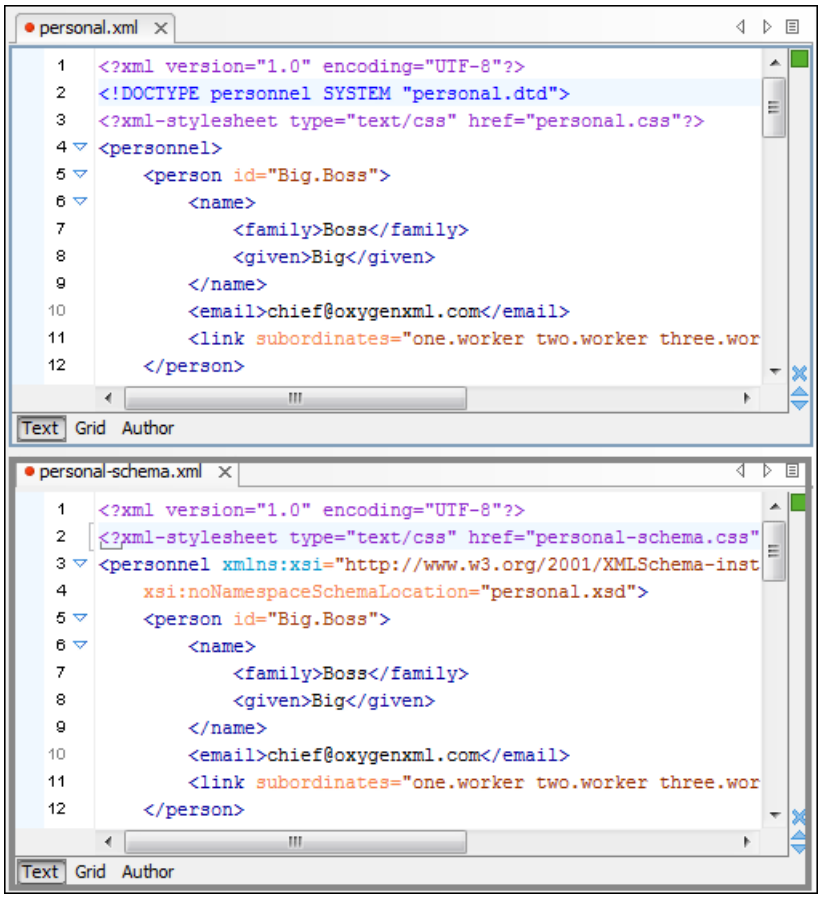


Figure 10: You can split the editing area by drag and drop of the editor title:

All the opened editors can be tiled horizontally/vertically or stacked together using actions from **Window** menu : **Tile Editors Horizontally,** **Tile Editors Vertically,** **Stack Editors.** When several tiled editors exist, and action **Synchronously scrolling** (from the same **Window** menu) is enabled, when scrolling inside one editor all other editors will also scroll.

Also the editing area can be divided vertically and horizontally with the split / unsplit actions available on the Split toolbar and the **Window** menu: **Split horizontally,** **Split vertically,** **Unsplit.**

The editor can be maximized or restored(same action as double click on the editor tab) by using the **Maximize/Restore Editor Area** action from the **Window** menu.

The tab strip is scroll-wheel sensitive, i. e. when there are more documents open than fit in the tab strip, the scroll wheel could be used to scroll left/right as is currently accomplished with the two arrows at the right. However that is not necessary for switching to other edited file as the following shortcuts can be used to display a small popup window that cycles through all opened files: Ctrl-F6 (Meta-F6 on Mac OS X) and Ctrl-Shift-F6 (Meta-Shift-F6 on Mac OS X).

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

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

Chapter 4

Editing Documents

Topics:

- *Working with Unicode*
- *Opening and Closing Documents*
- *Editing XML Documents*
- *Editing XML Schemas*
- *Editing Relax NG Schemas*
- *Editing NVDL Schemas*
- *Editing XSLT Stylesheets*
- *Editing XQuery Documents*
- *Editing CSS Stylesheets*
- *Editing XProc Scripts*
- *Editing Schematron Schemas*
- *SVG Documents*
- *Integrating External Tools*
- *Editing Very Large Documents*
- *Insufficient Memory*
- *Large File Viewer*
- *Hex Viewer*
- *Scratch Buffer*
- *Changing the User Interface Language*
- *Handling Read-Only Files*
- *Editing Documents with Long Lines*

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

As a Java application, Oxygen comes with a default Java input method for typing characters with Unicode codes. However, the default input method does not cover all the Unicode codes, for example the codes for some accented characters or characters found in East Asian languages. Such characters can be inserted in the editor panel of Oxygen either with *the **Character Map** dialog* available from menu **Edit** > **Insert from Character Map** or by installing a Java input method that supports the insertion of the needed characters. The *installation of a Java input method* depends on the platform on which Oxygen runs (Windows, Mac OS X, Linux, etc) and is the same for any Java application.

Note: Oxygen may not be able to display characters that are not supported by the operating system (either not installed or unavailable).

Tip: Windows XP/2003: You can enable 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

On loading documents of the type XML, XSL, XSD, and DTD, Oxygen reads the document prolog to determine the specified encoding type. This encoding is then used to instruct the Java Encoder to load support for and to save the document using the specified code chart. When the encoding type cannot be determined, Oxygen prompts and display the **Available Java Encodings** dialog which provides a list of all encodings supported by the Java platform.

If the opened document contains an unsupported character, Oxygen applies *the policy specified for handling such errors.* If the policy is set to REPORT, Oxygen displays an error dialog about the character not allowed by the encoding. If the policy is set to IGNORE, the character is removed from the document displayed in the editor panel. If the policy is set to REPLACE, the character is replaced with a standard replacement character for that encoding.

While in most cases you are using UTF-8, simply changing the encoding name causes the application to save the file using the new encoding.

On saving the edited document, if it contains characters not included in the encoding declared in the document prolog Oxygen detects the problem and signals it to the user. The user is responsible to resolve the conflict before saving the document.

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, 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 Oxygen when the document is edited and saved. This behavior can be changed in Oxygen from the *Encoding preferences panel*.

The Unicode Toolbar

The Unicode toolbar is switched on and off from **Window** > **Show Toolbar** > **Unicode** or from the contextual menu of the toolbar area and contains the actions **Change text orientation** with *the default shortcut* Ctrl + Shift + O and **Insert from Character Map**

The **Change text orientation** action enables editing documents in languages with right to left writing (Hebrew, Arabic, etc.). Please note that you may have to *set an appropriate Unicode aware font for the editor panel,* able to render the characters of the language of the edited file.

The **Insert from Character Map** action opens a dialog in which you can select one character in the matrix of all characters available in a font and insert it in the edited document. The action is available also in the **Edit** menu.

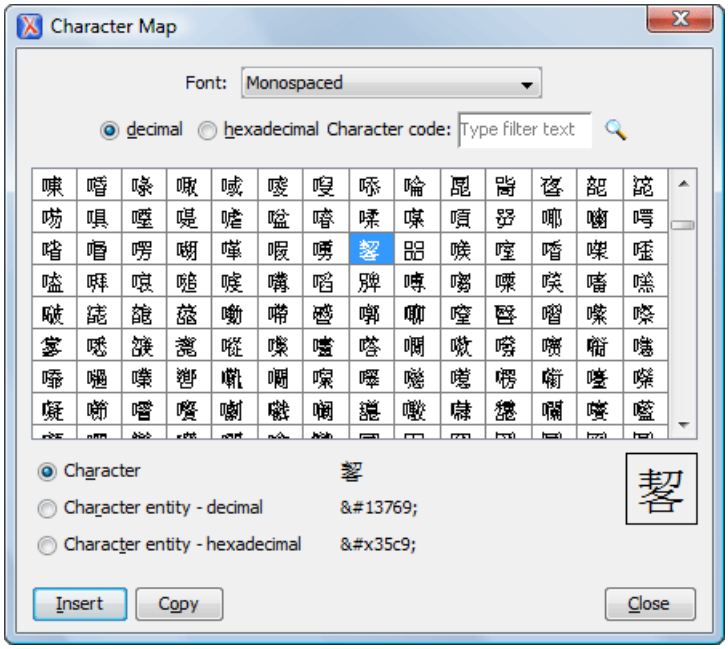


Figure 11: The Character Map dialog

The character selected in the character table or an entity with the decimal code or the hexadecimal code of that character can be inserted in the current editor. You will see it in the editor if *the editor font* is able to render it. The *Insert* button inserts the selected character in the editor. The *Copy* button copies it to the clipboard without inserting it in the editor.

A character can be located very quickly in the map if you know the Unicode code: just type the code in the search field above the character map and the character is selected automatically in the map. If the code is hexadecimal the radio button for hexadecimal codes is selected automatically. Selecting a radio button with the mouse starts searching the code in the map.

The *Character Map* dialog cannot be used to insert Unicode characters in *the grid version of a document editor*. Accordingly, the *Insert* button of the dialog will be disabled if the current document is edited in grid mode.

Opening and Closing Documents

This section explains the actions and wizards available for creating new files, opening existing files, and closing files.

Creating New Documents

This section details the procedures available for creating new documents.

The New Document Dialog

Oxygen supports many document types. This dialog presents the default associations between a file extension and the type of editor which opens the file for editing. You can override these default associations in the ***File Types*** *user preferences panel.*

1. Select **File** > **New (Ctrl+N)** or press the  **New** toolbar button.
   The New dialog is displayed. The supported document types are grouped into several categories:
   - **Recently used** - contains the list of most recently used files;
   - **New Document** - contains the list of supported document types (including, among others, XML, XSL, XML Schema, Document Type Definition, Relax NG Schema, XQuery, Web Services Definition Language, Schematron Schema, CSS File, Text File, PHP File, JavaScript File , Java File, C File, C++ File, Batch File, Shell File, Properties File, SQL File, XML Catalog, and PERL File).
   - **Global templates** - contains the list of predefined templates as well as templates defined in the *Document Templates* preferences page.
   - **Framework templates** - contains the list of templates defined in the *Document Type Association* preferences page, **Templates** tab.
2. Select a document type.
3. Click one of the following:
   - **Customize** - Action available only for XML, XML Schema, Schematron, and XSL file types. Depending on the document type, different properties can be set before creating the file.
   - **Create** - Uses default settings to create a file.

   If **Create** was clicked, the new file is created and opened in the editor view.
4. If **Customize** was clicked, the following dialog is opened. Depending on the selected document type, different properties can be set:

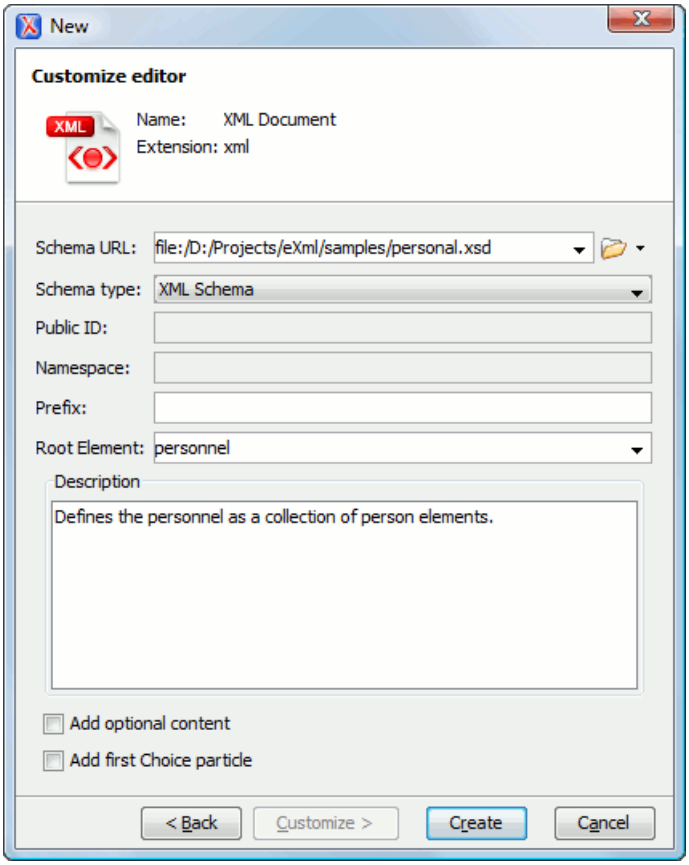


Figure 12: New XML Document Dialog

- **Schema URL** - Path to the schema file. When a file is selected, Oxygen analyzes its content and tries to fill-in the rest of the dialog;
- **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** - The document namespace;
- **Prefix** - The prefix for the namespace of the document root;
- **Root Element** - Populated with elements defined in the specified schema, enables selection of the element to be used as document root;
- **Description** - Shows a small description of the selected document root;
- **Add optional content** - When selected, the elements and attributes that are defined in the XML Schema as optional, are generated in the skeleton XML document;
- **Add first Choice particle** - When selected, the first element of an *xs:choice* schema element is generated in the skeleton XML document created in a new editor panel when the **OK** button is pressed.

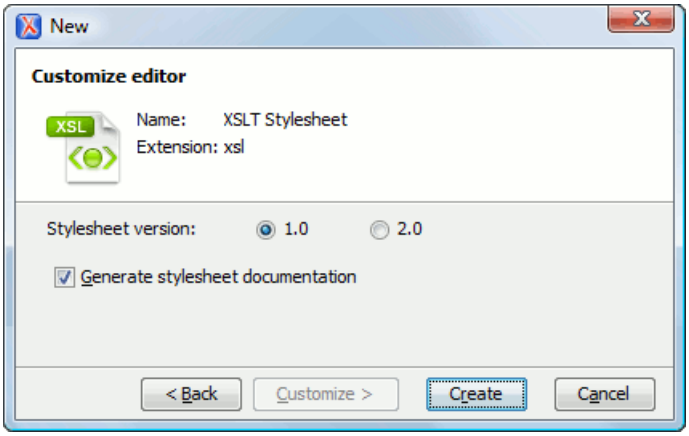
- 

Figure 13: New XSL Document Dialog

- **Stylesheet version** - Stylesheet version number. Possible options: 1.0 and 2.0;
- **Generate stylesheet documentation** - Generates the stylesheet documentation.

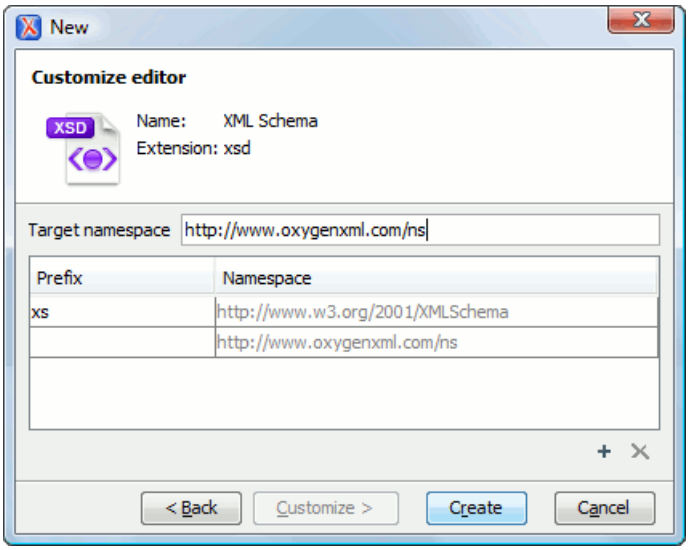
- 

Figure 14: New XML Schema Document Dialog

- **Target namespace** - Specifies the schema target namespace;
- **Namespace prefix declaration table** - Contains namespace prefix declarations. Table information can be managed using the **New** and **Delete** buttons.

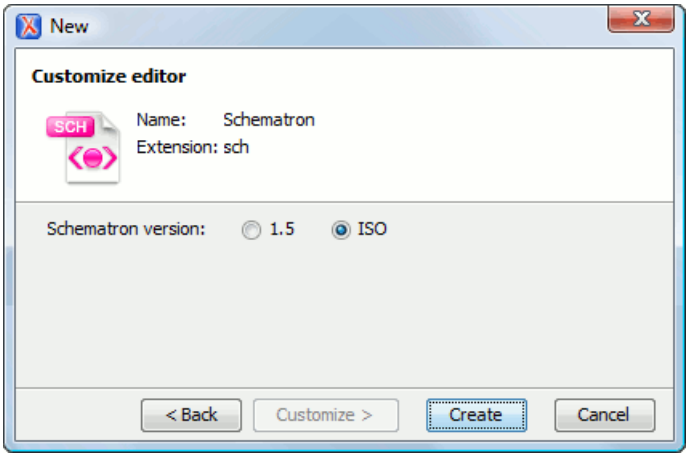
- 

Figure 15: New Schematron Document Dialog

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

5. Press **Create** to create the file.

Creating Documents Based on Templates

*The **New** wizard* enables you to select predefined templates or templates that have already been 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 install directory. Also in the option page **Options** > **Preferences** > **Editor** > **Templates** > **Document Templates** can be specified a custom templates folder to be scanned.

1. Go to menu **File** > **New** .
2. Select a document type.
3. Press the **Finish** button.

The newly created document already contains the structure and content provided in the template.

Document Templates

Templates are documents containing a predefined structure. They provide starting points on which one can rapidly build new documents that repeat the same basic characteristics: file type, prolog, root element, existing content. Oxygen installs a rich set of templates for a number of XML applications. You may also create your own templates from **Options** > **Preferences** > **Editor** > **Templates** > **Document Templates** and share them with other users.

You can also use *editor variables* in the template files' content and they will be expanded when the files are opened.

Saving Documents

The edited document can be saved with one of the following actions:

- **File** > **Save** > **(Ctrl+S)** .
- The **Save** toolbar button. If the document was not saved yet it displays the **Save As** dialog.
- **File** > **Save As** : displays the **Save As** dialog, used either to name and save an open document to a file or to save an existing file with a new name.
- **File** > **Save To URL** displays the **Save to URL** dialog, used either to name and save an open document to a file or to save an existing file with a new name, *using FTP/SFTP/WebDAV*.

- **File** > **Save All** : Saves all open documents. If any document does not have a file, displays the **Save As** dialog.

Opening Existing Documents

Documents can be opened using one of the following actions:

- Go to menu **File** > **Open (Ctrl+O)** to display the **Open** dialog. The start folder of the **Open** dialog can be either the last folder visited by this dialog or the folder of the currently edited file. This can be *configured in the user preferences.*
- Press the  **Open** toolbar button to display the same dialog.
- Go to menu **File** > **Open URL ...** to open a document *using FTP/SFTP/WebDAV*.
- Press the  **Open URL ...** toolbar button to run the same action.
- Go to menu **File** > **Open/Find Resource ... (Ctrl+Shift+R)** to look for a document from *the current project* or from a DITA map opened in *the DITA Maps Manager view* by typing a part of the file name.
- Press the  **Open/Find Resource ...** toolbar button to run the same action.
- Go to menu **File** > **Revert** to load the last saved file content. All unsaved modifications are lost.
- Go to menu **File** > **Reopen** to reopen one of the recently opened document files. The list containing recently opened files can be emptied by invoking the **Clear history** action.
- Select the action **Open** from the **Project** view contextual menu. This opens the selected file from the **Project** view.
- Start the application from the command line with the paths of one or more local files as parameters. The specified files will be opened automatically when the application is started:
  - scriptName [pathToXMLFile1] [pathToXMLFile2] ... where scriptName is the name of the startup script for your platform (oxygen.bat on Windows, `oxygen.sh` on Unix/Linux, `oxygenMac.sh` on Mac OS) and *`pathToXMLFileN`* is the name of a local XML file
  - an XML file and a schema file to be associated automatically to the file and used for validation and content completion:

    ```
    scriptName -instance pathToXMLFile -schema pathToSchemaFile -schemaType
    XML_SCHEMA|DTD_SCHEMA|RNG_SCHEMA|RNC_SCHEMA -dtName documentTypeName
    ```

    where `scriptName` is the name of the startup script for your platform (oxygen.bat on Windows, oxygen.sh on Unix/Linux, oxygenMac.sh on Mac OS), pathToXMLFile is the name of a local XML file, pathToSchemaFile is the name of the schema which you want to associate to the XML file, the four constants (XML_SCHEMA, DTD_SCHEMA, RNG_SCHEMA, RNC_SCHEMA) are the possible schema types (W3C XML Schema, DTD, Relax NG schema in full syntax, Relax NG schema in compact syntax). The next parameter, documentTypeName, specifies the name of the *Document Type* for which the schema is defined. If the Document Type is already set in the options pages, then its schema and type will be updated.

  The two possibilities of opening files at startup by specifying them in the command line are explained also if the startup script receives one of the *-h* or *--help* parameters.

Also when the Tree Editor perspective is activated the current document in the Editor perspective will be opened and displayed as a tree of XML elements.

Open/Find Resource

The **Open/Find Resource** dialog is opened from menu **File** > **Open/Find Resource ... (Ctrl+Shift+R)** or from the toolbar button  **Open/Find Resource ...**. It allows quickly finding a file in the current Oxygen project or in one of the DITA maps opened in *the **DITA Maps Manager** view* by typing only a few letters of the file name.

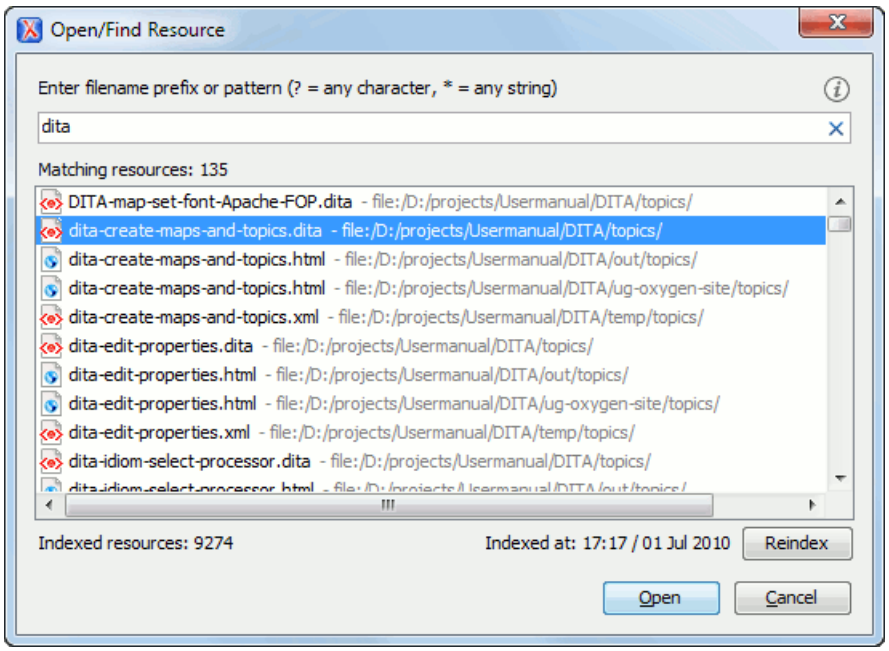


Figure 16: The Open/Find Resource Dialog

The list of file names that match the file pattern typed in the search field is updated automatically when a character is inserted or deleted in the search field. For each matching file, the full path name is displayed in the list.

Because this operation involves extensive access to the hard drive, a caching mechanism is used to gather the paths of all files linked in the current project. When the first search is performed, all project files are indexed and added to the cache. The next search operations use the information extracted from the cache, thus improving the processing time. The cache is kept for the currently loaded project only, so when you perform a search in a new project the cache is rewritten. Also, the cache is reset when you press the **Reindex** button.

If there is no file found that matches your file pattern, a possible cause is that the file was added to the Oxygen project after the last caching operation. In this case re-indexing the project files from the **Reindex** button will enable the file to be found. The date and time of the last index operation is displayed below the file list.

Once you find the files that you want to open, select them in the list and press the **Open** button. Each of the selected files is opened in *the editor associated with the type of the file*.

Opening and Saving Remote Documents via FTP/SFTP/WebDAV

Oxygen supports editing remote files, using the FTP, SFTP and WebDAV protocols. The remote files can be edited exactly as the local ones, for example they can be added to a project, and can be subject to XSL and FO transformations.

You can open one or more remote files in *the dialog **Open using FTP/SFTP/WebDAV*** .

A WebDAV resource can be locked when it is opened in Oxygen Oxygen XML Author by checking the option *Lock WebDAV files on open* to prevent other users to modify it concurrently on the server. If a user tries to edit a locked file, the application will display a error message that contains the lock owner's name. The lock is released automatically when the editor for that resource is closed in Oxygen Oxygen XML Author .

To improve the transfer speed, the content exchanged between Oxygen Oxygen XML Author 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 Dialog Open Using FTP/SFTP/WebDAV

The dialog **Open using FTP/SFTP/WebDAV** is displayed from the menu **File** > **Open URL ...** or from the toolbar button **Open URL ...**.

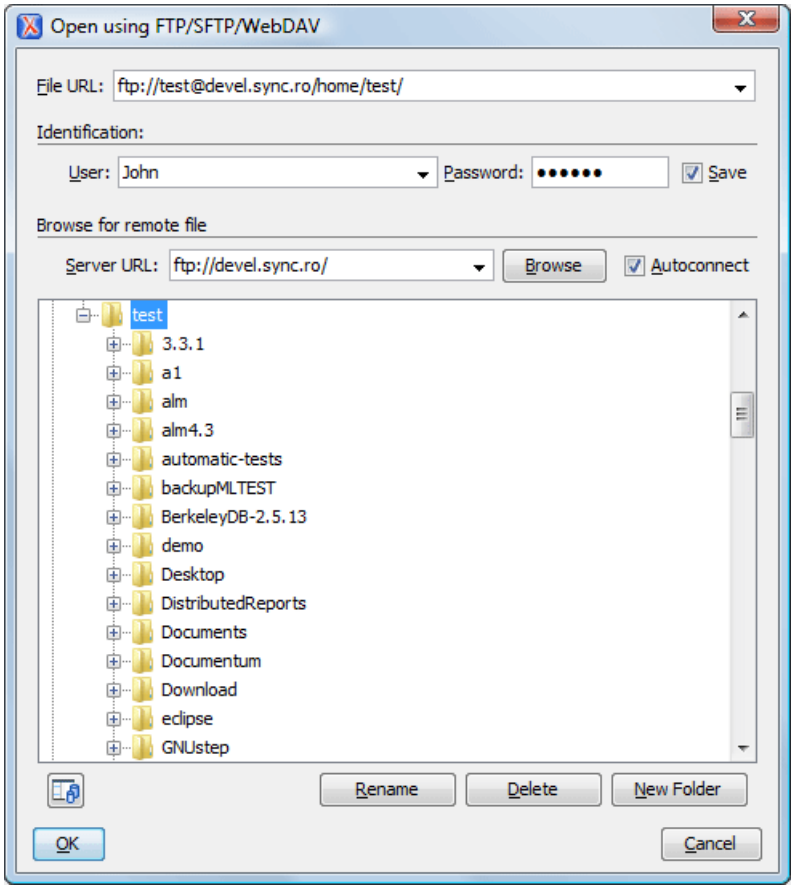


Figure 17: Open URL dialog

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 http://some.site/test.xml, in case the file is accessible through normal HTTP protocol, or ftp://anonymous@some.site/home/test.xml if the file is accessible through anonymous FTP.

  This combo box is also displaying the current selection when the user changes selection by browsing the tree of folders and files on the server.

- 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. Into the server combo it may be specified the protocol (HTTP, HTTPS or FTP), the name or IP of the server and, in case of WebDAV, the path to a WebDAV directory.

  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/.

  When accessing a WebDAV server, along with the protocol and the host, it must be specified also the directory of the WebDAV repository.

  Important:

  Make sure that the repository directory ends in a slash "/".

  Ex: https://www.some-webdav-server.com:443/webdav-repository/, http://devel:9090/webdav/

  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/WebDAV 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 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 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 . This means that Oxygen 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.

How to Add a HTTPS Server Certificate to Oxygen

You add a HTTPS server certificate to the Java key store by exporting it to a local file using any HTTPS-capable Web browser (for example Internet Explorer) and then importing this file into the JRE using the **keytool** executable bundled with the JRE. The steps are the following using Internet Explorer (if you use other browser the procedure is similar):

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.

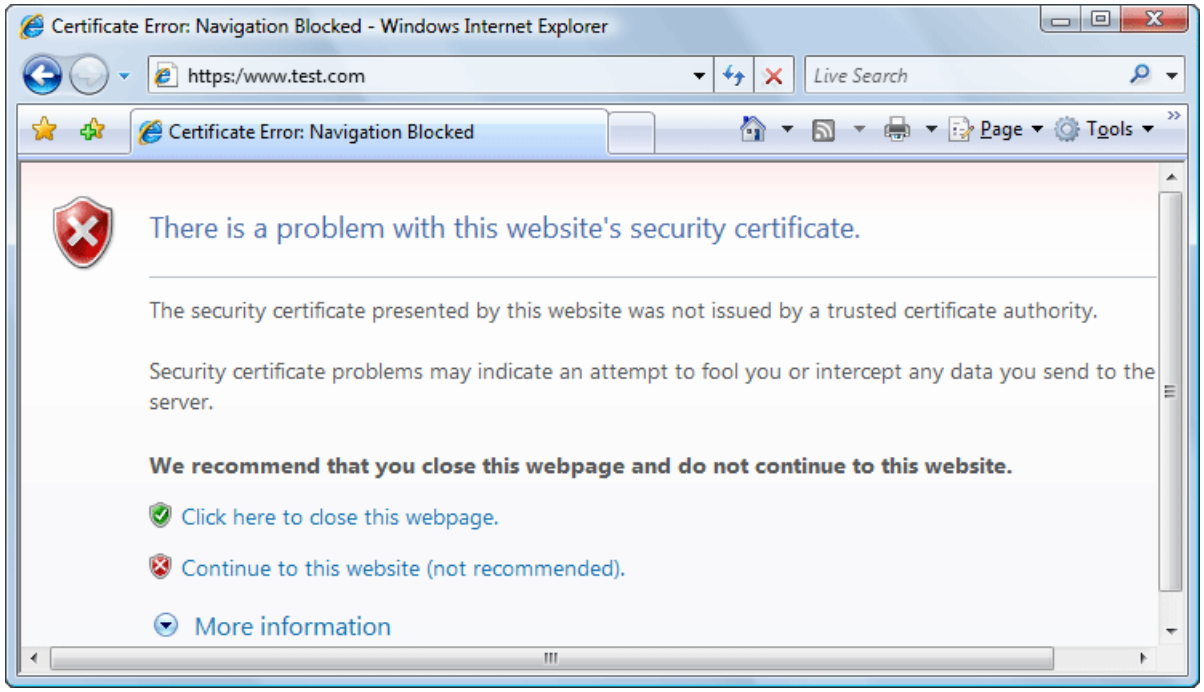


Figure 18: Security alert - untrusted certificate

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 .

a) Open a text-mode console.
b) Go to the `lib/security` subfolder of your JRE directory, that is of the directory where it is installed the JRE running Oxygen . You find the home folder of the JRE in the *java.home* property that is displayed in the About dialog, the **System properties** tab.
c) Run the following command:

```
..\..\bin\keytool.exe -import -trustcacerts -file server.cer -keystore
cacerts
```

The `local-file.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. As a workaround you can delete the `cacerts` file, re-type the command and enter as password any combination of at least 6 characters. This will set the password for future operations with the key store.

3. Restart Oxygen .

Opening the Current Document in a Web Browser

To open the current document in your computer's default Web browser, use the **Open in browser** action available on the **Document** > **File** menu and also on the **Document** toolbar. This is useful to see the effect of applying an XSLT stylesheet or a CSS stylesheet on a document which specifies the stylesheet using an *xml-stylesheet* processing instruction.

Closing Documents

To close documents use one of the following methods:

- Go to menu **File** > **Close (Ctrl+W)** : Closes only the selected tab. All other tab instances remain opened.
- Go to menu **File** > **Close All** : Closes all open documents. If a document is modified or has no file, a prompt to save, not to save, or cancel the save operation is displayed.
- Select the item **Close** from the contextual menu of an editor tab: Closes the selected editor.
- Select the item **Close Other Files** from the contextual menu of an editor tab: Closes the other files except the selected tab.
- Select the item **Close All** from the contextual menu of an editor tab: Closes all open editors within the panel.

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
- document's total number of characters
- line width
- indent with tabs state
- indent size

The view can be accessed from **Window** > **Show View** > **Properties** > **Editor properties**

To copy a value from the **Properties** view in the clipboard, for example the full file path, use the **Copy** action available on the contextual menu of the view.

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 the Oxygen application.

Supported Schema Types for XML Documents

The supported schema types are:

- W3C XML Schema (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

The default schema used by *content completion* is the schema of the document type that matches the edited document. *The list of document types* available at **Options** > **Preferences** > **Document Type Association** contains a set of rules for associating a schema with the current document when no schema is explicitly specified within the document. The schema has one of following the 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 you can observe that the list of tags to be inserted is changing.

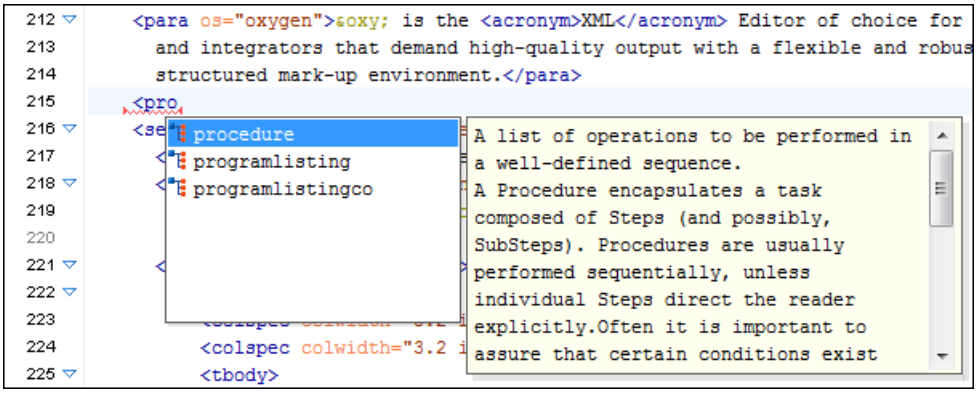


Figure 19: Content Completion Driven by DocBook DTD

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:

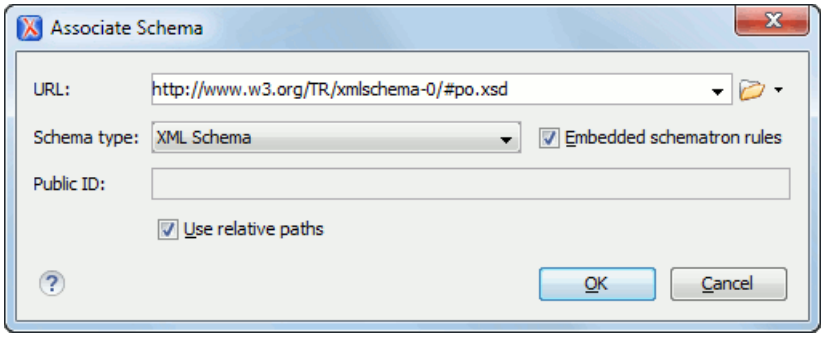


Figure 20: The Associate Schema Dialog

The following options are available:

- **URL** - contains a predefined set of schemas that are used more often and it also keeps a history of the last used schemas. The URL must point to the schema file which can be loaded from the local disk or from a remote server through HTTP(S), FTP(S) or a *custom protocol*.
- **Schema type** - selected automatically from the list of possible types in the **Schema type** combo box (XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, NVDL) based on the extension of the schema file that was entered in the **URL** field.
- **Public ID** - Specify a public ID if you have selected a DTD.
- **Embedded schematron rules** - if you have selected XML Schema or Relax NG schemas with embedded Schematron rules, enable this option.
- **Use relative paths** - 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.

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/nvdl/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 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.

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 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 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 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 menu **Document** > **XML Document** > **Learn Structure (Ctrl+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 menu **Document** > **XML Document** > **Save Structure (Ctrl+Shift+S)** . Enter the DTD file path.
4. Press the *Save* button.

Streamline with Content Completion

Oxygen's intelligent Content Completion feature enables rapid, in-line identification and insertion of structured language elements, attributes and in some cases their parameter options.

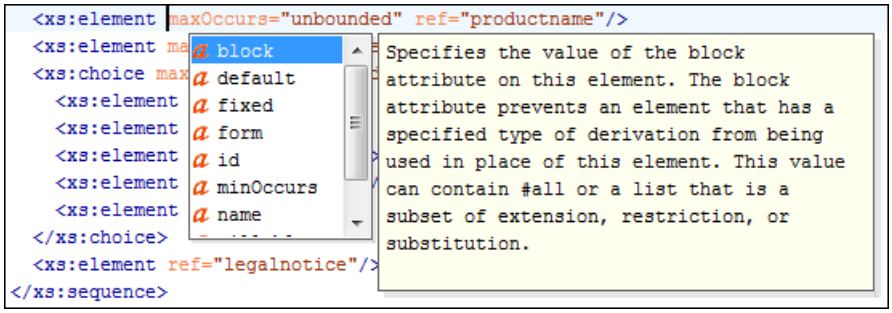


Figure 21: Content Completion Assistant

Oxygen logs the URL of the detected schema in the *Information view*.

If the Content Completion assistant is *enabled in user preferences* (the option **Use Content Completion**), then it is displayed:

- automatically, after a configurable delay from the last key press of the < character. The delay is *configurable in Preferences* as a number of milliseconds from last key press.
- on demand, by pressing CTRL+Space on a partial element or attribute name.

Elements are highlighted in the list using the Up and Down cursor keys. Here are the options to insert the selected content:

- press the Enter key or the Tab key to insert both the start and end tags.
- press CTRL + Enter. The application inserts both the start and end tags, separated by an empty line. The cursor is positioned on the empty line on an indented position with regard to the start tag.

Note: When the DTD, XML Schema or RELAX NG schema specifies required child elements for the newly added element, they will be inserted automatically only if the Add Element Content option (found in **Preferences** > **Editor** > **Content Completion** options 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, for the element if these two options are enabled.

After inserting the element, the cursor will be 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 will display the Content Completion list once again. This time it will contain the list of allowed attribute names. If the attribute supports a fixed set of parameters, the assistant list will display the list of valid parameters. If the parameter setting is user-defined and therefore variable, the assistant will be closed to enable manual insertion. The values of the attributes can be learned from the same elements in the current document.
- after the > char of the start tag if the element has no attributes.

The content assistant can be started at any time by pressing CTRL+Space Also it can be started with the action **Start Content Completion** (default shortcut is CTRL + Slash) which *can be configured in* ***Preferences*** > ***Menu Shortcut Keys*** : category *Content Completion*, description *Start Content Completion*. The effect is that the context-sensitive list of proposals will be shown in the caret's current position if element, attribute or attribute value insertion makes sense. 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 are dependent on the element structure specified in the DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema associated to the edited document.

The number and type of elements displayed by the 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. All elements that can't be child elements of the current element according to the specified schema are not displayed.

If the schema for the edited document defines attributes of type ID and IDREF the content assistant will display for IDREF attributes a list of all the ID values already present in the document for an easy insertion of a valid ID value at the cursor position in the document. This is available for documents that use DTD, XML Schema and Relax NG schema.

Also values of all the *xml:id* attributes are treated as ID attributes and 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 window.

The operation of the Content Completion assistant is configured by the options available in the options group called *Content Completion*.

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 reads the beginning of the document and resolves the location of the DTD, XML Schema, Relax NG schema, or NVDL schema;

  Note:

  Limitation: In case of XML Schema, the content completion takes into account only the schema declarations from the document's root element. If a schema declaration is attached to other element of the XML document, then it will be ignored.

- 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 *Content Completion options*.Oxygen 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 *Content Completion options*.Oxygen 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:

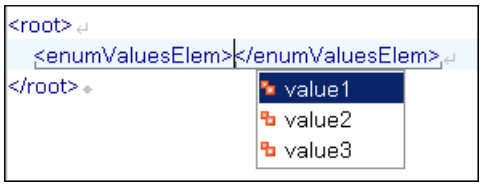


Figure 22: Content Completion assistant - element values in Relax NG documents

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. The tooltip window can be invoked at any time using the F2 shortcut.

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 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 functions in the attributes *path*, *select*, *context*, *subject*, *test* depending on *the Schematron options* that are set in Preferences. If the Saxon 6.5.5namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.3.0.5 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:

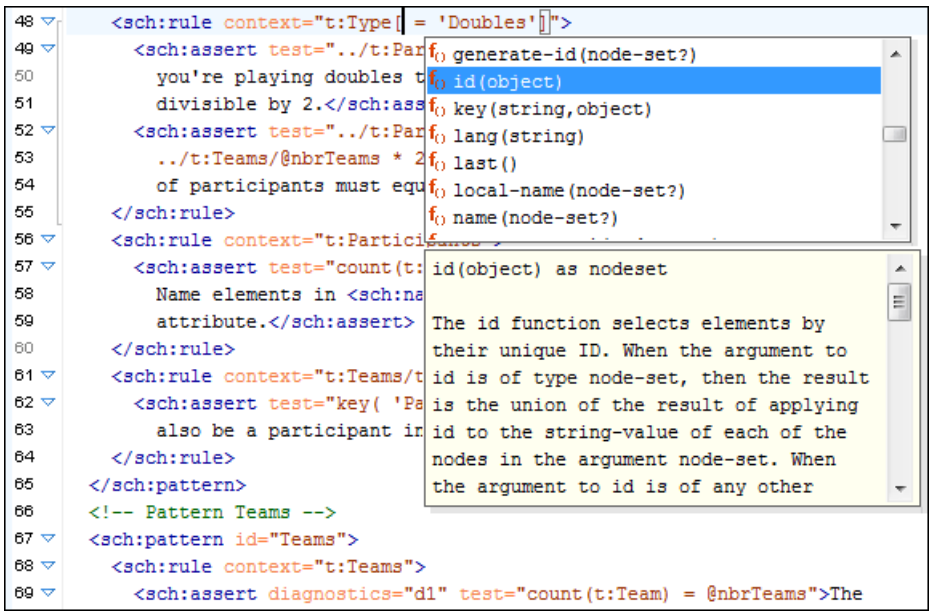


Figure 23: XSLT extension functions in Schematron schemas documents

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

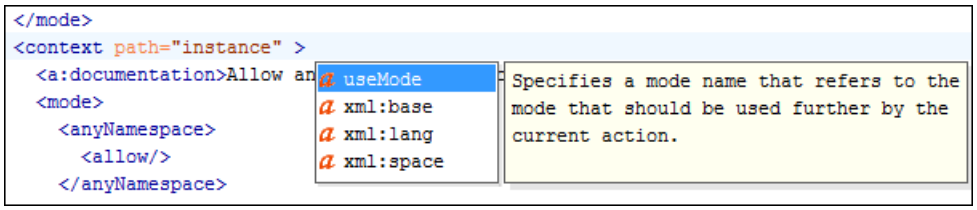


Figure 24: Schema annotations displayed at Content Completion

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 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 Panels

Information about the current element being edited is also available in the Model panel and Attributes panel, located on the left-hand side of the main window. The Model panel and the Attributes panel combined with the powerful Outline view provide spatial and insight information on the edited document.

The Model Panel

The Model panel presents the structure of the current edited tag and tag documentation defined as annotation in the schema of the current document.

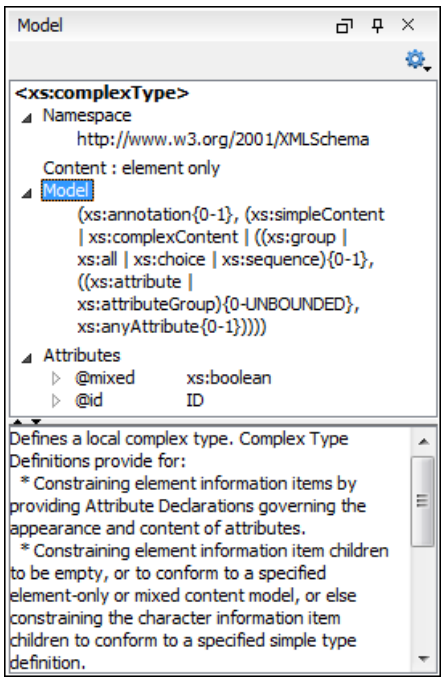


Figure 25: The Model View

The Model panel is comprised of:

- *An element structure panel.*
- *An annotation panel.*

The Element Structure Panel

The element structure panel shows the structure of the current edited or selected tag in a tree-like format.

The information includes the name, model and attributes the currently edited tag may have. The allowed attributes are shown along with imposed restrictions, if any.

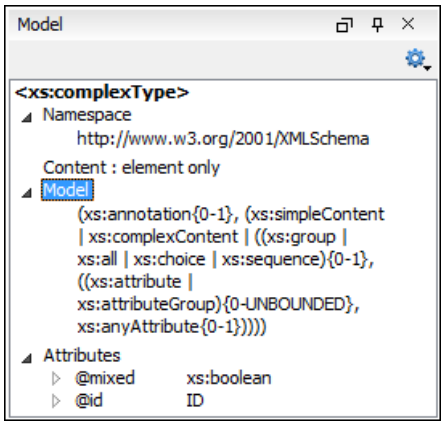


Figure 26: The Element Structure Panel

The Annotation Panel

The Annotation panel displays the annotations that are present in the used schema for the currently edited or selected tag. This information can be very useful to developers learning XML because it has small available definitions for each used tag.

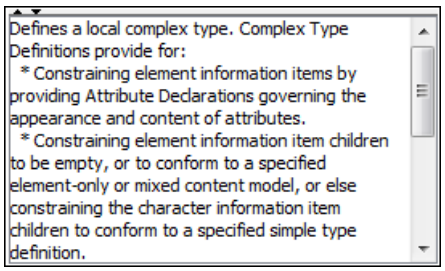


Figure 27: The Annotation panel

The Attributes Panel

The Attributes panel presents all possible attributes of the current element and allows to insert attributes in the current element or change the value of the attributes already used in the element. The attributes already present in the document are painted with a bold font. Clicking on the Value column of a table row will start editing the value of the attribute from the selected row. If the possible values of the attribute are specified as list in the schema associated with the edited document, the Value column works as a combo box where you can select one of the possible values to be inserted in the document.

The Attributes table is sortable, three sorting orders being available by clicking on the columns' names. Thus the table's contents can be sorted in ascending order, in descending order or in a custom order, where the already used attributes are placed 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.

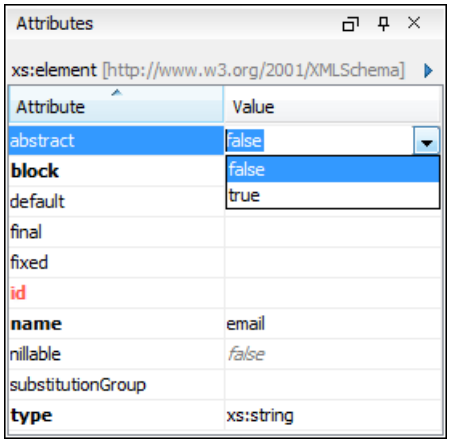


Figure 28: The Attributes Panel

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 will insert 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.

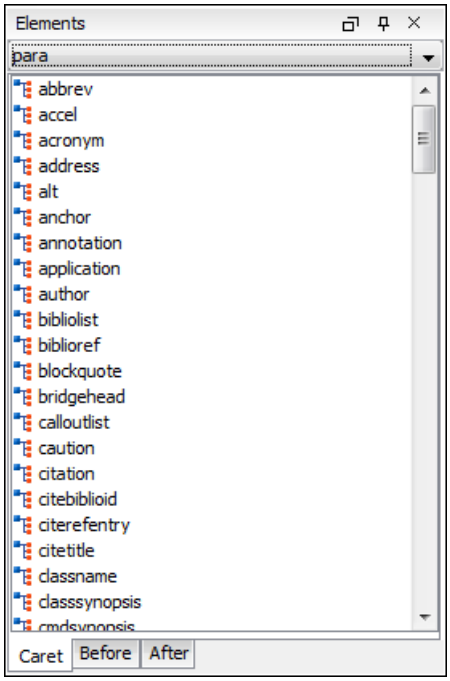


Figure 29: The Elements View

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.

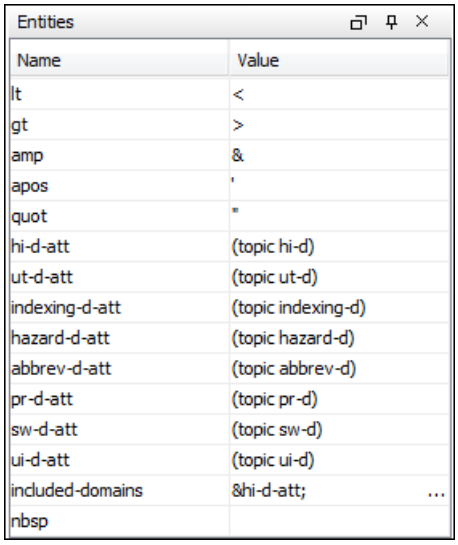


Figure 30: The Entities View

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 if the word at cursor position is a prefix of such a short name. If there is no prefix at cursor position, that is the character at the left of cursor is a whitespace, all the code templates are listed.

Oxygen comes with a lot of predefined code templates but you can *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 can use the Content Completion on request shortcut key (usually CTRL-SPACE) or the Code Templates on request shortcut key (CTRL-SHIFT-SPACE). 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 following variables can appear in a code template:

- *${caret}* - The caret position after inserting the code template.
- *${selection}* - The position of the current selection in the inserted template.
- *${env(ENV_VAR_NAME)}* - The value of the environment variable *ENV_VAR_NAME*.
- *${system(var.name)}* - The value of the system variable *var.name*.
- *${date(yyyy-MM-dd)}* - A date in the format: 4 digits for year, 2 digits for month, 2 digits for day.

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 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.

If you select menu **Document** > **Validate** > **Check Well-Formedness(Ctrl+Shift+W)** or click the toolbar button **Check Well-Formedness**Oxygen checks if your document is *Namespace Well-Formed XML*. 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 : **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 Oxygen **Validate document** 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, it is possible to select the validation phase.

Marking Validation Errors and Warnings

A line with a validation error or warning will be marked in the editor panel by underlining the error region with a red color. Also a red sign will mark the position in the document of that line on the right side ruler of the editor panel. The same will happen for a validation warning, only the color will be yellow instead of red.

The ruler on the right side of the document is designed to display the errors and warnings found during the validation process and also to help the user to locate them more easily. The ruler contains the following areas:

- Top area containing a success validation indicator that will turn green in case the validation succeeded or red otherwise.

  A more detailed report of the errors is displayed in the tooltip of the validation indicator. In case there are errors, only the first three of them will be presented in the tooltip.
- Middle area where the error markers are depicted in red (with a darker color tone for the current selected one). The number of markers shown can be limited by modifying the setting **Options** > **Preferences** > **Editor** > **Document checking** > **Maximum number of problems reported per document** .

  Clicking on a marker will highlight the corresponding text area in the editor. The error message is displayed both in the tool tip and in the error area on the bottom of the editor panel.

  The *Document checking user preferences* are easily accessible from the button displayed at the beginning of the error message on the bottom of the editor panel.
- Bottom area containing two navigation arrows that will go to the next or to the previous error and a button for clearing all the error markers from the ruler. The same actions can be triggered from menu **Document** > **Automatic validation** > **Next Error (Ctrl + .)** and **Document** > **Automatic validation** > **Previous Error (Ctrl + ,)** .

The validation status area is the line at the bottom of the editor panel that presents the message of the current validation error selected on the right side ruler. Clicking on  opens the *document checking* page in Oxygen user preferences.

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

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.

Caching the Schema Used for Validation

If you don't change the active editor and you don't switch to other application, the schema associated to the current document is parsed and cached by the first **Validate Document** action and is reused by the next actions without re-parsing it. This increases the speed of the validate actions if the schema is large or is located on a remote server on the Web. To reset the cache and re-parse the schema you have to use the **Reset Cache and Validate** action. This action will also re-parse the catalogs and reset the schema used for content completion.

Automatic Validation

Oxygen *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.*

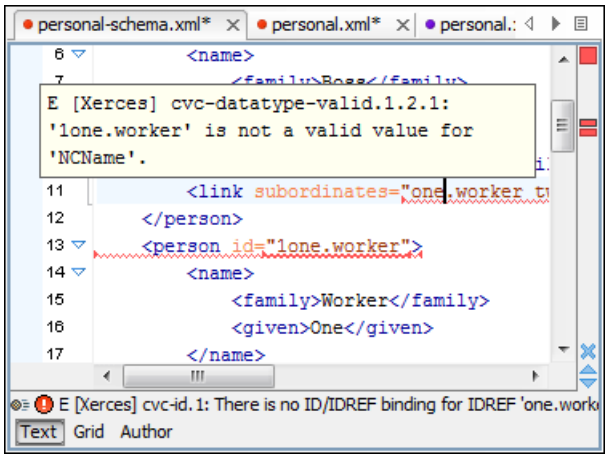


Figure 31: Automatic Validation on the Edited Document

If the error message is long and it is not displayed completely in the error line at the bottom of the editing area, double-clicking on the error icon at the left of the error line or on the error line displays an information dialog with the full error message. The arrow buttons of the dialog enable the navigation to other errors issued by the Automatic Validation feature.

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 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 for linked messages:

- **LIBXML** - Included in Oxygen (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 (--xinclude) 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, the parameter --dtdvalid ${ds} must be added manually to the DTD validation command line. ${ds} represents the detected DTD declaration in the XML document.

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 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 on validation against a W3C XML Schema like:

Unimplemented block at ... xmlschema.c

These errors can be avoided by specifying the full path to the LIBXML executable file.

- **Saxon SA** - Included in Oxygen. 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 specification or according to the W3C XML Schema 1.1 specification. This can be *configured in Preferences*.
- **MSXML 4.0** - Included in Oxygen (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 (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. 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* 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. 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.

A custom validator cannot be applied on files loaded through an *Oxygen custom protocol plugin* developed independently and added to Oxygen after installation.

Linked Output Messages of an External Engine

Validation engines display messages in an output view at the bottom of the Oxygen window. If such an output message (warning, error, fatal error, etc) spans between three to five 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.
- Description: message content (the string *Description:* followed by the content of the message that will be displayed in the output view).

Validation Scenario

A complex XML document is usually split in smaller interrelated modules which do not make much sense individually and which cannot be validated in isolation due to interdependencies with the other modules. A mechanism is needed to set the main module of the document which in fact must be validated when an imported module needs to be 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 a XSLT template with the name `chunk` which is called by `chunk-code.xsl`. The parameters defined in `param.xsl` are used in the other modules without being redefined.

Validation of `chunk-code.xsl` as an individual XSLT stylesheet issues a lot of misleading errors referring to parameters and templates used but undefined which are only caused by ignoring the context in which this module is used in real XSLT transformations and in which it should be validated. To properly validate such a module, a validation scenario must be defined to set the main module of the stylesheet and also the validation engine used to find the errors. Usually this is the engine which applies the transformation in order to detect in validation the same errors that would be issued by transformation.

A second benefit of a validation scenario is that the stylesheet can be validated with several engines to make sure that it can be used in different environments with the same results. For example an XSLT stylesheet needs to be 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 (Saxon 9) or any connection to a database that supports validation (Berkeley DB XML Database, eXist XML Database, Software AG Tamino, Documentum xDb (X-Hive/DB) 10 XML Database) can be set as validation engine.
- An XML document in which the master file includes smaller fragment files using XML entity references.

How to Create a Validation Scenario

Follow these steps for creating a validation scenario:

1. Open the **Configure Validation Scenario** dialog from menu **Document** > **Validate** > **Configure Validation Scenario** or from the **Validate** toolbar.
   The following dialog is displayed. It contains the following types of scenarios:
   - **Default validation** scenario validates the input using Oxygen default validation options that apply to the type of the current document;
   - **Predefined** scenarios are organized in categories depending on the type of file they apply to and can be easily identified by a yellow key icon that marks them as *read-only*. If the predefined scenario is the framework's default scenario its name is written in bold font. If you try to edit one of these scenarios, Oxygen 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.

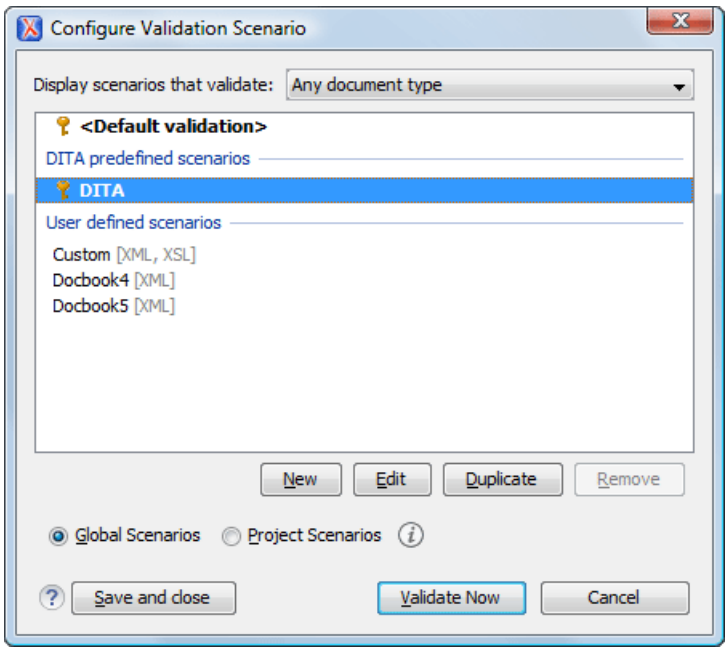


Figure 32: Configure Validation Scenario

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.

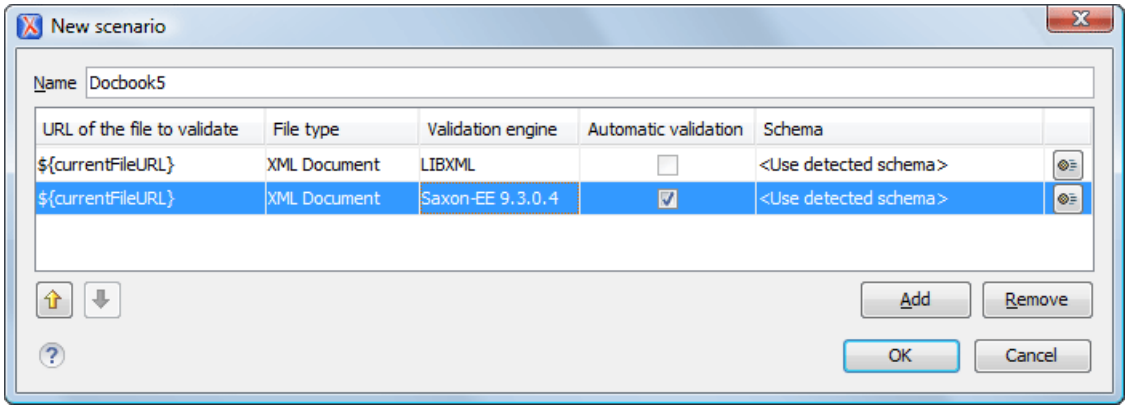


Figure 33: Add / Edit a Validation Unit

The table holds the following information:

- **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 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 for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the validation is done by the default engine set in Preferences pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, etc) 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** - Active when you set the **File type** to **XML Document**.

- **Settings** - Contains an action that allows you to set a schema, when validating XML documents, or a list of extensions when validating XSL or XQuery documents.

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:

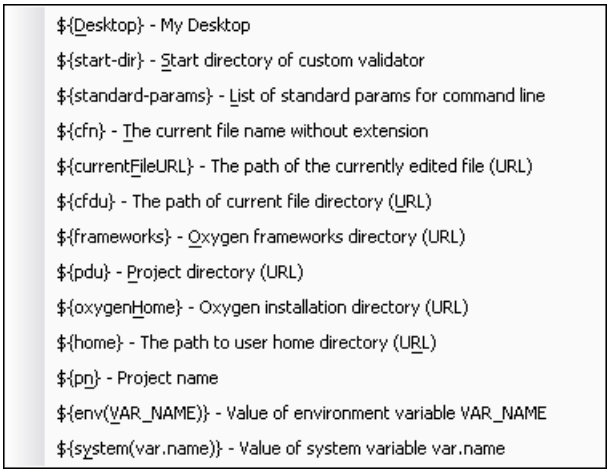


Figure 34: Insert an Editor Variable

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

Sharing Validation Scenarios

Sometimes a group of users want to apply the same validation settings like the main module where the validation starts, the validation engine, the schema, extensions of the engine. In order to apply the same settings consistently it is preferable to share the validation scenario with the settings by storing it at project level and sharing the project file using a source versioning system (CVS, SVN, Source Safe, etc).

You specify that you want to store a scenario at project level by selecting the option **Project Scenarios** instead of the default option **Global Scenarios**.

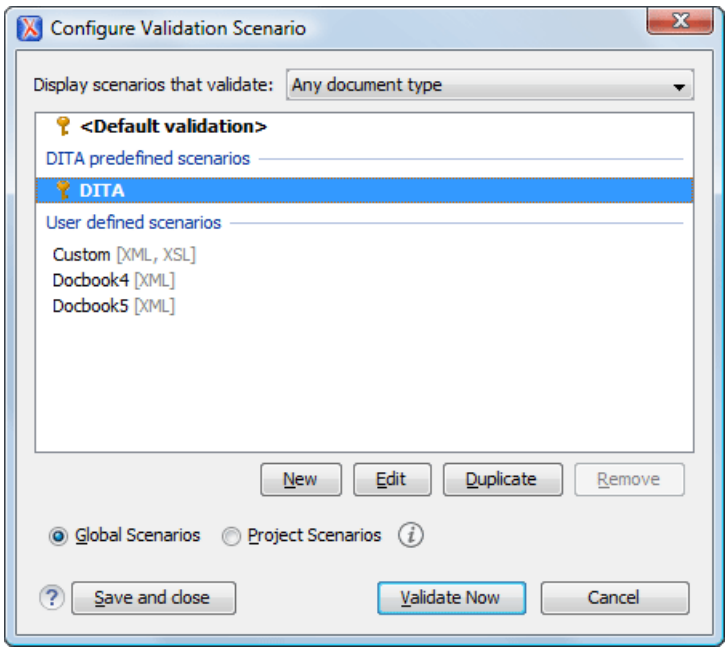


Figure 35: Configure Validation Scenario

The option **Global Scenarios** ensures that the scenarios are saved in the user home directory. After changing the selection to **Project Scenarios** the scenario list will be stored in the project file. If your project is saved on a source versioning system then your team can use the scenarios you defined.

Validation Actions in the User Interface

Use one of the actions for validating the current document:

- Select menu **Document** > **Validate** > **Validate Document (Ctrl+Shift+V)** or click the button **Validate Document** available in the **Validate** toolbar. This action returns an error list in the message panel. Mark-up of current document is checked to conform with the specified DTD, XML Schema or Relax NG schema rules. It caches the schema and the next execution of the action uses the cached schema.
- Select menu **Document** > **Validate** > **Reset Cache and Validate** or click the button **Reset Cache and Validate** available in the **Validate** toolbar to reset the cache with the schema and validate the document. This action also parses again the XML catalogs and reset the schema used for content completion. It returns an error list in the message panel. Mark-up of current document is checked to conform with the specified DTD, XML Schema or Relax NG schema rules.
- Select menu **Document** > **Validate** > **Validate with (Ctrl+Shift+H)** or click the button **Validate with** available in the **Validate** toolbar. This action can be used to validate the current document using a selectable schema (XML Schema, DTD, Relax NG, NVDL, Schematron schema). It returns an error list in the message panel. Mark-up of current document is checked to conform with the specified schema rules. The **Validate with** action does not work for files loaded through an *Oxygen custom protocol plugin* developed independently and added to Oxygen after installation.
- Select menu **Document** > **Schema** > **Open External Schema** or click the button **Open External Schema** available in the **Document** toolbar to open the schema used for validating the current document in a new editor.
- Select submenu **Validate Selection** > **Validate** in the contextual menu of **Project** panel, to validate all selected files with their declared schemas.
- Select submenu **Validate Selection with Schema ...** > **Validate With ...** of the contextual menu of **Project** panel, to select a schema and validate all selected files with that schema.
- Select the submenu **Validate** > **Configure Validation Scenario ...** of the contextual menu of **Project** panel, to configure and apply a validation scenario in one action to all the selected files in the **Project** panel, .

The button **Validation options** available on the **Validate** toolbar allows quick access to the *validation options* of the built-in validator in the Oxygen user preferences page.

Also you can select several files in the **Project** panel 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 the **Project** view.

If there are too many validation errors and the validation process takes too long, you can *limit the maximum number of reported errors in Preferences*.

References to XML Schema Specification

If validation is done against XML Schema Oxygen indicates a specification reference relevant for each validation error. The error messages contain an **Info** field that when clicked will open the browser on the *XML Schema Part 1:Structures* specification at exactly the point where the error is described. This allows you to understand the reason for that error.

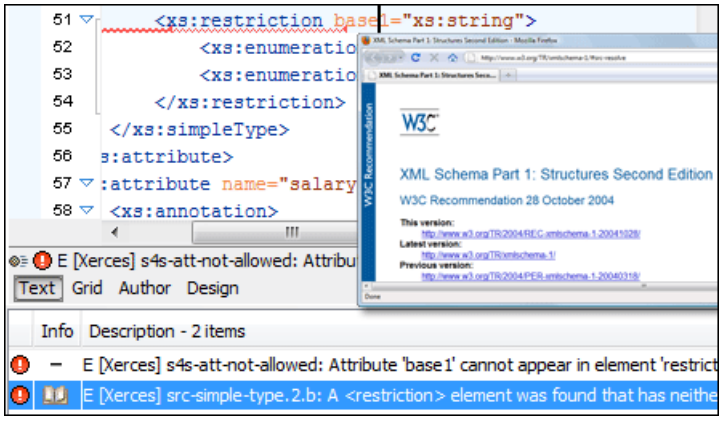


Figure 36: Link to Specification for XML Schema Errors

Resolving References to Remote Schemas with an XML Catalog

When a reference to a remote schema must be used in the validated XML document for interoperability purposes, but a local copy of the schema should be actually used for validation for performance reasons, the reference can be resolved to the local copy of the schema with an *XML catalog*. For example, if the XML document contains a reference to a remote schema `docbook.rng` like this:

```
<?oxygen RNGSchema="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng"
    type="xml"?>
```

it can be resolved to a local copy with a catalog entry:

```
<system systemId="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:

```
<system systemId="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1"
    uri="topic.xsd"/>
```

Document Navigation

This section explains various methods for navigating the edited XML document.

Quick Document Browsing Using Bookmarks

The bookmark concept is the same as in other IDEs: you can mark a position in an edited document so that you can return after further editing and browsing through one or more documents opened at the same time. Up to nine distinct bookmarks can be placed in any opened document. Configurable shortcut key strokes are available to place bookmarks and to return to any of the marked positions.

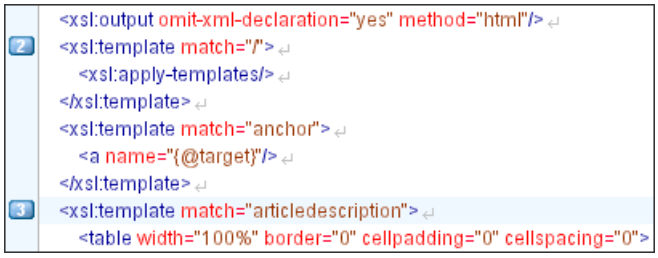


Figure 37: Editor Bookmarks

The key strokes *can be configured* from **Options** > **Preferences** ->Menu shortcut keys.

A bookmark can be placed from **Edit** > **Bookmarks** ->Create, from **(F9)** > **Edit** > **Bookmarks** > **Bookmarks Quick Creation (F9)** , by clicking the toolbar button **Bookmarks Quick Creation** and by clicking in the margin of the editing area, to the left of the line number area, reserved for bookmarks.

Quickly switching to a position marked by a bookmark can be done by **Edit** > **Bookmarks** ->**Go to**.

Two contextual menu actions are available:

- **Remove** - Removes the current bookmark;
- **Remove all** - Removes all bookmarks set in the 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 38: Folding of the XML Elements

An unique feature of Oxygen is the fact that the folds are persistent: the next time you will open the document the folds are restored to the last state so you won't have to collapse the uninteresting parts again.

To toggle the folded state of an element click on the special mark displayed in the left part of the document editor next to the start tag of that element or click on the action **Toggle fold** available from the contextual menu or from the menu **Document** > **Folding** > **Toggle fold** The element extent is marked with a grey line displayed in the left part of the edited document. The grey line always covers the lines of text comprised between the start tag and end tag of the element where the cursor is positioned.

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

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

Outline View

The Outline view offers the following functionality:

- *XML Document Overview* on page 85
- *Outline Specific Actions* on page 86
- *Modification Follow-up* on page 86
- *Document Structure Change* on page 86
- *Document Tag Selection* on page 87

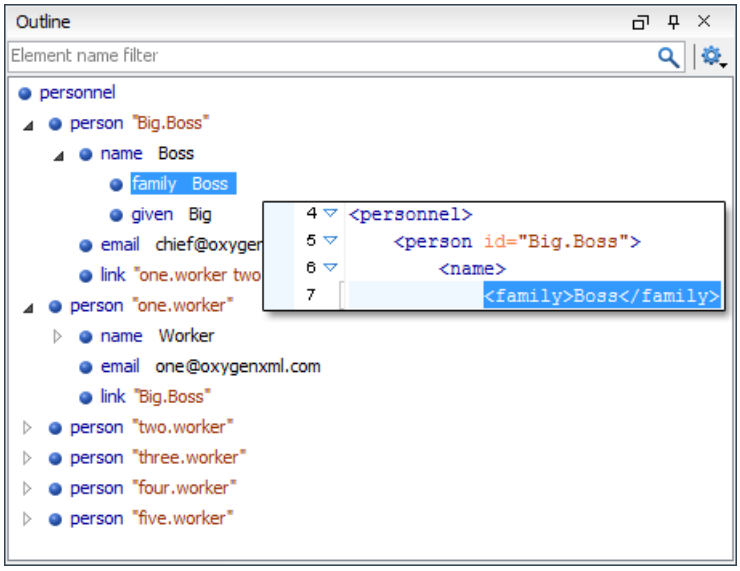


Figure 39: The Outline View

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. That makes easier for the user to be aware of the document structure and the way 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 *Expand more* and *Collapse all* items of the popup menu available on the Outline tree enlarge or reduce the set of nodes of the edited document currently visible in the view. The tree expansion action is a faster alternative to mouse clicks on the plus signs of the tree when one wants to access quickly a node deeply nested in the hierarchy of document nodes. When a large number of nodes become expanded and the document structure is not clear any more, the collapsing action clears the view quickly by reducing the depth of the expanded nodes to only one child of the currently selected node.

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. An easy-to-spot exclamation mark sign is used as element icon, a

red underline decorates the element name and value and a tooltip provides more information about the nature of the error.

Outline Specific Actions

The following actions are available in the **Settings** menu on the Outline view's toolbar:

- **Selection update on caret move** - Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram will be also selected in the **Outline** view.
- **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 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 editing, the Outline view dynamically follows the modifications introduced by the user, showing in the middle of the panel the node which is currently being modified. This gives the user better insight on location where in the document one is positioned and how the structure of the document is affected by one's modifications.

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)** 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.

Document Tag Selection

The Outline view can also be used to search for a specific tag's location and contents in the edited document. Intuitively, by selecting with the left mouse button the desired tag in the Outline view, the document is scrolled to the position of the selected tag. Moreover, the tag's contents are selected in the document, making it easy to notice the part of the document contained by that specific tag and furthermore to easily copy and paste the tag's contents in other parts of the document or in other documents.

You can double click the tag in the Outliner tree to move focus to the editor.

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

Navigation Buttons

These buttons are available in the editor's main toolbar:

- **Go to Last Modification** : Moves the cursor to the last modification in any opened document.
- **Back** :Moves the cursor to the previous position.
- **Forward** :Moves the cursor to the next position. Enabled after at least one press of the **Back** button.

Using the Go To Dialog

The *Go to* dialog available from **Find** > **Go to ... (Ctrl+L (Cmd+L on Mac))** enables you to go to a precise location in the current edited file specified by line and column or by an offset relative to the beginning of the file.

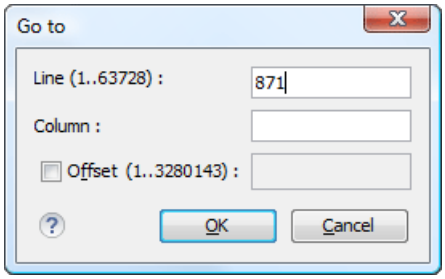


Figure 40: Go to Dialog

Complete the dialog as follows:

- **Line** - destination line in the current document;
- **Column** - destination column in the current document;
- **Offset** - destination offset relative to the beginning of document.

Grouping Documents in XML Projects

This section explains how to create and work with projects in the Oxygen application.

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 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">
%xinclude;
]>
<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"?>
<?oxygen RNGSchema="test.rng" type="xml"?>
<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"?>
<?oxygen RNGSchema="test.rng" type="xml"?>
<test>
    <a xml:id="a1" xml:base="a.xml">test</a>
</test>
```

The XInclude support in Oxygen is turned on by default. You can *toggle it* by going to the entry **Enable XInclude processing** in the menu **Options** > **Preferences ...** > **XML** > **XML Parser** . When enabled, Oxygen will be able to validate and transform documents comprised of parts added using XInclude.

Using the Project View

The Project view is designed to assist the user in organizing and managing related files grouped in the same XML project. The actions available on the context menu and toolbar associated to this panel, enable the creation of XML projects and shortcuts to various operations on the project documents.

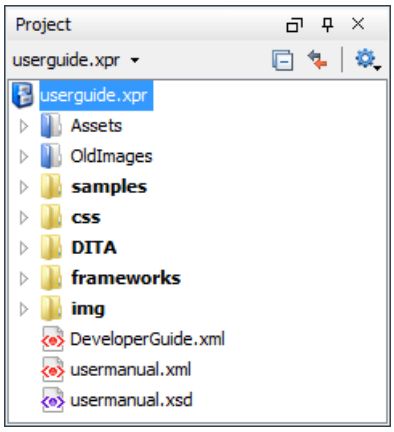


Figure 41: The Project View

The default layout initialized by the **Window** > **Reset Layout** menu item positions the **Project** view on the left side of the Oxygen window, above *the Outline view.* A closed view can be quickly reopened at any time with the **Project** > **Show Project View** menu action.

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

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

The following actions are grouped in the upper right corner:

- **Collapse All** - Collapses all project tree folders. You can also collapse/expand a project tree folder if you select it and press the **Enter** key.
- **Link with editor** - When selected, the project tree highlights the currently edited file.
- **Settings** - A submenu containing the following actions:
  - **Filters** - Allows you to filter the information displayed in the **Project** view. Click the toolbar button to set filter patterns for the files you want to show or hide. Also, you can set filter patterns for the linked directories that are hidden.
  - **Show Full Path** - When selected, linked files and folders are presented with a full file path.

The files are organized in an XML project usually as a collection of folders. There are two types of folders:

- *Logical folders* - marked with a blue icon on Windows and Unix/Linux ( ) and a magenta icon on Mac OS X ( ) and do not have any connection with folders on the disk. This folder type has no correspondent on the physical disk, being used as containers for related items. Creating and deleting them in Oxygen does not affect the file system on disk.
- *Linked folders* - marked with a yellow icon on Windows and a blue icon on Mac OS X which is exactly the folder icon used by the Windows Explorer and Mac OS Finder applications. They content mirror a real folder existing in the file system on disk. They can be

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 File System** - Replicates the structure of physical folders on disk. The newly created logical folder contains the file structure of the folder it points to.
- **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.

Add Content to a Logical Folder

You can add content to a logical folder using one of the actions available in the contextual menu:

- **Add Folder** - Adds a link to a physical folder, whose name and content mirror a real folder 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.
- **Add Files** - Adds links to files on disk.
- **Add Edited File** - Adds a link to the current edited file to the project.

Managing Project Content

You can create linked folders by dragging and dropping a folder from the Windows Explorer / Mac OS X Finder over the project tree or by selecting in the contextual menu **Add Folder** . Also the structure of the project tree can be changed with drag and drop operations on the files and folders of the tree.

When adding files to a project, the default target is the project root. To change a target, select a new folder. Files may have multiple instances within the folder system, but cannot appear twice within the same folder.

To remove one or more files or folders, select them in the project tree and press the **Delete** key or run the contextual menu action **Remove from Project**. To remove a file or folder both from project and disk, run the contextual menu action **Remove from Disk (Shift+Delete)** which is available for both logic and linked files.

Caution: In most cases this action is irreversible, deleting the file permanently. Under particular circumstances (if you are running a Windows installation of Oxygen and the *Recycle Bin* is active) the file is moved to *Recycle Bin*.

To create a file inside a linked folder, choose the **New** > **File** action from the contextual menu.

There are three ways yo rename an item in the **Project** view: To begin editing an item name in the **Project** view, select the item and do one of the following:

- invoke the **Rename** action from the contextual menu;
- press **F2**;
- click the selected item.

To finish editing the item name press **Enter**.

Note:

- Files or folders are renamed both in the Oxygen **Project** view and on the local disk;

- The **Rename** action is also available on logic files.

If a project folder contains many documents, a certain document can be quickly located in the project tree if the user selects with the mouse the folder containing the desired document (or some arbitrary document in this folder) and types the first characters of the document name. The desired document is automatically selected as soon as the typed characters uniquely identify its name in the folder. The selected document can be opened by pressing the **Enter** key, by double-clicking it and with one of the **Open** actions from the pop-up menu. To open a file of known type with other editor than the default one, use the **Open with** action.

The project file is saved automatically on disk, every time the content of the Project view is modified by actions like adding or removing files or folders and drag and drop to/from the Project view.

Validate Files

The currently selected files in the **Project** 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:

- **Check Well-Formedness** - checks if the selected file or files are well-formed.
- **Validate** - validates the selected file or files against their associated schema. EPUB files make an exception, because this action triggers a ***Validate and Check for Completeness*** operation.
- **Validate with Schema...** - validates the selected file of files against a specified schema.
- **Configure Validation Scenario** - allows you to configure and run a *validation scenario*.

Applying Transformation Scenarios

The currently selected files in the **Project** view can be transformed in one step with one of the actions **Transform** > **Apply Transformation Scenario** , **Transform** > **Configure Transformation Scenario ...** and ***Transform*** > ***Transform with...*** available on the right-click menu of the **Project** 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

A list of useful file properties like the ones available in *the **Properties** view* can be obtained with the **Properties** action of the contextual menu invoked on a file node of the **Project** view tree, in the following dialog:

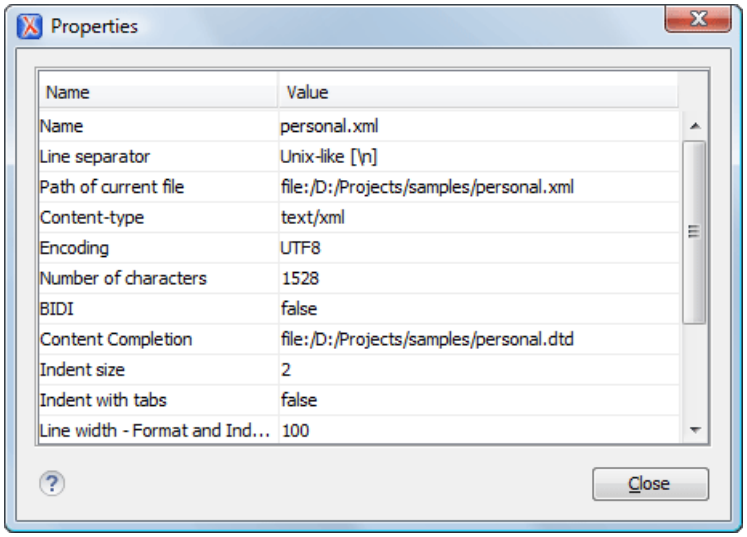


Figure 42: The Properties Dialog

Right-clicking any object in the tree view displays the **Project** menu with functions that can be performed on, or from the selected object. Options available from the **Project** menu are specific to the object type selected in the tree view.

You can also use drag and drop to arrange the files in logical folders (but not in linked 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)

- **Show in Explorer** (or **Show in Finder** on Mac OS X) - Opens an OS-specific finder/explorer window, with the file or folder in question selected in the finder/explorer window.
- **Open with** - Open selected file with one of internal tools: *SVG Viewer*, *Hex Viewer*, *Large File Viewer*, *MathML Editor*, WSDL/SOAP Analyzer, *DITA Maps Manager*, *Archive Browser*.
- **Open All Files** - Action available only when at least one folder is selected. Opens in the editor view all files contained by the selected resources.
-  **Find/Replace in Files** - Allows you to *find and replace text in multiple files*.
-  **Check Spelling in Files** - Allows you to *check the spelling of multiple files*.
-  **Open in SVN Client** - *Syncro SVN Client* tool is opened and it highlights the selected resource in its corresponding working copy.

Menu Level Actions

The following actions are available in the **Project** menu:

-  **New Project** - Creates a new, empty project.
-  **Open Project ... (Ctrl+F2)** - Opens an existing project. An alternate way to open a project is to drop an Oxygen XPR project file from the file explorer in the **Project panel**.
- **Save Project As...** - Allows you to save the current project under a different name.
- **Validate all project files** - Checks if the project files are well-formed and their mark-up conforms with the specified DTD, XML Schema, or Relax NG schema rules. It returns an error list in the message panel.
- **Show Project View** - Displays the project view.
- **Reopen Project** - Contains a list of links of previously used projects. This list can be emptied by invoking the **Clear history** action.

Team Collaboration - Subversion

There is a *SVN (Subversion) Client* application embedded in Oxygen. You may start it from the **Tools** menu and use it for synchronizing your working copy with a central repository.

Another way of starting it is by using the contextual menu of the **Project** tree: **Team** > **Open in SVN Client** . This action displays the Syncro SVN Client and shows the selected project file in the **Working Copy** view.

Project Level Settings

You can store into the project not only lists of files and directories, but also transformation scenarios and other setting specific to that project. For more information see the *Preference Sharing* and *Sharing the Transformation Scenarios* topics.

Working with XML Catalogs

When Internet access is not available or the Internet connection is slow the *OASIS XML catalogs* present in the list *maintained in the XML Catalog Preferences panel* will be scanned trying to map a remote system ID (at document validation) or a URI reference (at document transformation) pointing to a resource on a remote Web server to a local copy of the same resource. If a match is found then Oxygen will use the local copy of the resource instead of the remote one. This enables the XML author to work on his/hers XML project without Internet access or when the connection is slow and waiting until the remote resource is accessed and fetched becomes unacceptable. Also *XML catalogs* make

documents machine independent so that they can be shared by many developers by modifying only the XML catalog mappings related to the shared documents.

Oxygen supports any XML catalog file that conforms to one of:

- the *OASIS XML Catalogs Committee Specification v1.1*
- the *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*.

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">
```

Inside an XML Schema if an *xs:import* statement specifies only the *namespace* attribute, without the *schemaLocation* attribute, Oxygen will try to resolve the specified namespace URI through one of the XML catalogs configured in Preferences pages.

The URN can be resolved to a local schema file with a catalog entry like:

```
<system systemId="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1"
    uri="topic.xsd"/>
```

An XML Catalog file can be created quickly in Oxygen 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*.

User preferences related to XML Catalogs can be configured from **Options** > **Preferences ...** > **XML** > **XML Catalog**

XML Catalog

An XML catalog helps the XML parser to check a document for errors if the schema or a part of the schema is not available, for example when an Internet connection is not available.

👉 **Important:**

Oxygen XML Editor collects all the catalog files listed in the installed frameworks. No matter what the Document Type Association matches the edited file, all the catalog mappings are considered when resolving external references.

👉 **Important:**

The catalog files settings are available for all editing modes, not only for the **Author** mode.

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 will generate an approximation of the source schema.

The conversion functionality is available from **Tools** > **Generate/Convert Schema... (Ctrl+Alt+T)** , from the **Project** view contextual menu - the action **Open with** > **Generate/Convert Schema** and from the toolbar button **Generate/Convert Schema...** .

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.

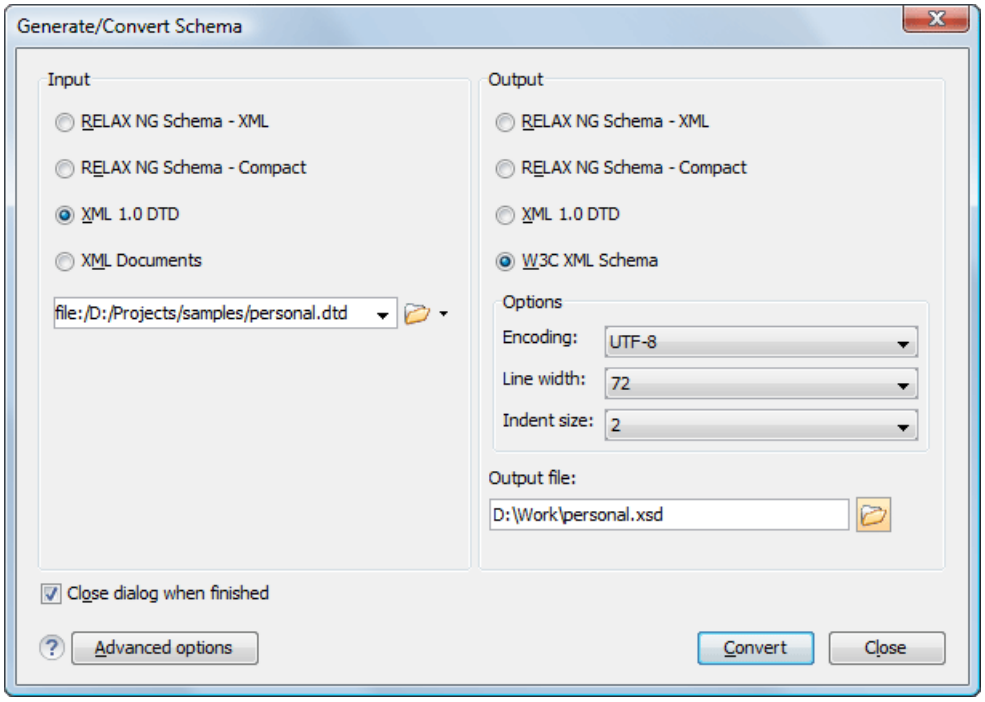


Figure 43: Convert a Schema to Other Schema Language

The language of the source schema is specified with one of the four radio buttons of the **Input** panel. If the **XML Documents** button is selected more than one file selection is allowed in the list below the group of radio buttons in case the conversion is based on a set of XML files instead of a single file.

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:

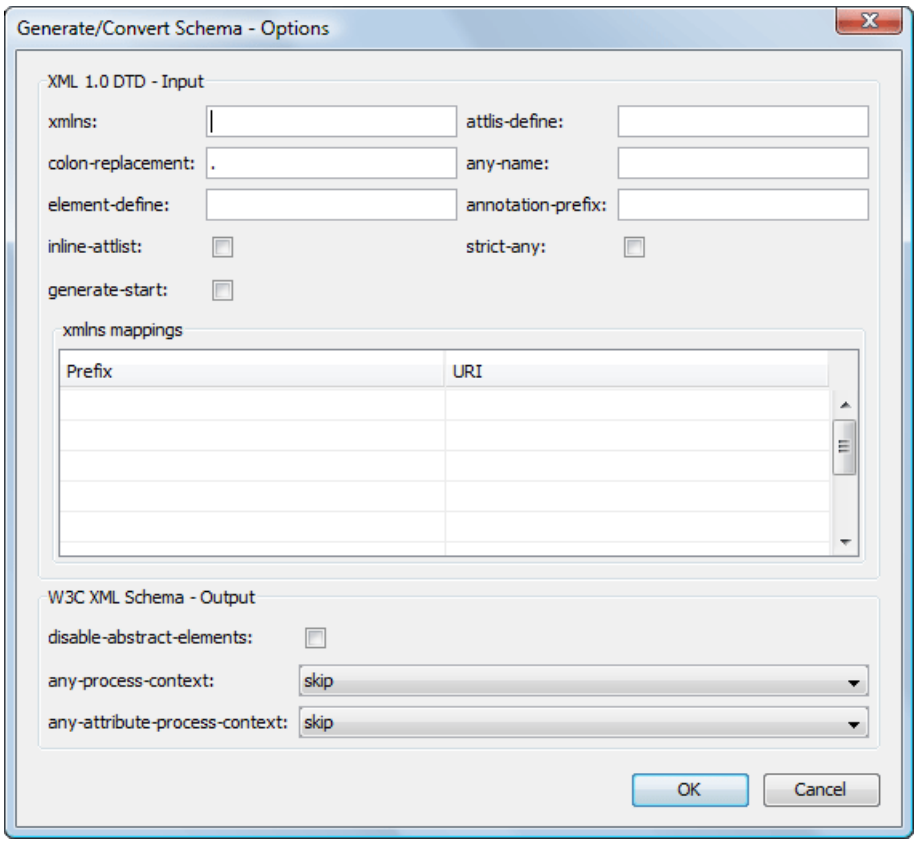


Figure 44: Convert a Schema to Other Schema Language - Advanced Options

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).

Editing XML Tree Nodes

A *Well-Formed XML document* can be viewed and edited in Oxygen also as a tree of XML elements. This is possible in the Tree Editor perspective, available from **Tools** > **Tree Editor (Ctrl+T)** . The Tree Editor provides specially designed views, toolbars and an editable tree allowing you to execute common tree actions like create/delete nodes, edit node names, move nodes with drag and drop.

If you want to be able to edit XML documents that are not well-formed and still have a tree view of the document you should use the *Outline view* in the Editor perspective.

Formatting and Indenting 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 formatting of just one XML element see the action ***Pretty print element*** . 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.

Viewing Status Information

Status information generated by the **Schema Detection**, **Validation**, **Automatic validation** and **Transformation** threads are fed into the **Information** view allowing the user to monitor how the operation is being executed.

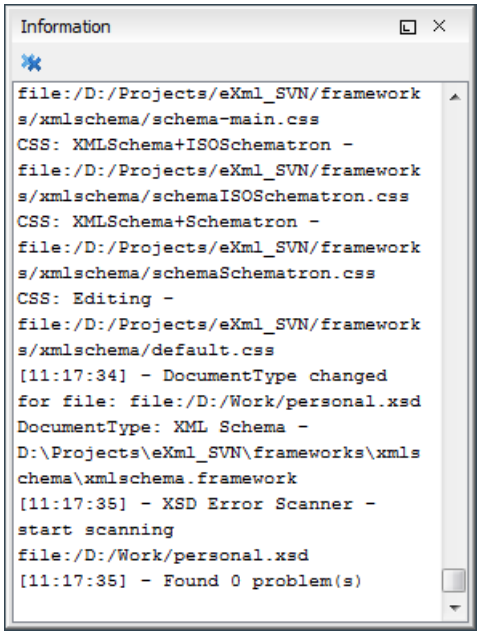


Figure 45: Information view messages

Messages contain a timestamp, the name of the thread that generated it and the actual status information. The number of displayed messages can be controlled from the ***Options** panel*.

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

Image Preview

Images and SVG files from the **Project** view can be previewed in a separate panel.

To preview an image you have to either double click the image name or click the **Preview** action from the **Project**'s tree contextual menu. Supported image types are GIF, JPEG/JPG, PNG, BMP. Once the image is displayed in the **Preview** panel using the actions from the contextual menu one can scale the image at its original size (1:1 action) or scale it down to fit in the view's available area (**Scale to fit** action).

To preview a *SVG file* click the **Preview** action from the **Project**'s tree contextual menu. Once the SVG is displayed in the **Preview** panel the following actions are available on the contextual menu: **Zoom in**, **Zoom out**, **Rotate** and **Refresh**.

Making a Persistent Copy of Results

The **Results** panel displays the results from the following operations:

- *document validation*
- *checking the form of documents*
- *XSLT or FO transformation*
- *find all occurrences of a string in a file*
- *find all occurrences of a string in multiple files*
- *applying an XPath expression to the current document*

To make a persistent copy of the **Results** panel use one of the actions:

- **File** > **Save Results** - displays the **Save Results** dialog, used to save the result list of the current message tab. The action is also available on the right click menu of the **Results** panel.
- **File** > **Print Results** - displays the **Page Setup** dialog used to define the page size and orientation properties for printing the result list of the current **Results panel**. The action is also available on the right click menu of the **Results** panel.
- **Save Results as XML** on the contextual menu - saves the content of the **Results** panel in an XML file with the format:

```
<Report>
    <Incident>
        <engine>The engine who provide the error.<engine>
        <severity>The severity level<severity>
        <Description>Description of output message.</Description>
        <SystemID>The location of the file linked to the message.</SystemID>
        <Location>
           <start>
                <line>Start line number in file.<line>
                <column>Start column number in file<column>
           </start>
           <end>
                <line>End line number in file.<line>
                <column>End column number in file<column>
           </start>
        </Location>
    </Incident>
</Report>
```

Locking and Unlocking XML Markup

For documents with fixed markup such as forms in which the XML tags are not allowed to be modified but only their text content, editing of the XML tag names can be disabled and re-enabled with the action available from **Document** > **Source** > **Locks / Unlocks the XML Tags** or from the toolbar button **Locks / Unlocks the XML tags**

There is a default lock state for all opened editors in the *Preferences XML Editor Format* page.

Adjusting the Transparency of XML Markup

Most of the time you want the content of a document displayed on screen with zero transparency. When you want to focus your attention only on editing text content inside XML tags Oxygen offers the option of reducing the visibility of the tags by increasing their transparency when they are displayed. There are two levels of tag transparency: semi-transparent markup and transparent markup. For the opposite case, when you want to focus on the tag names, the text transparency can be set to one of two levels: semi-transparent text and transparent text. To change the level of transparency click the toolbar button **Adjust Contrast** available on the **Edit** toolbar.

Note: On Windows XP and Windows Vista, depending on antialiasing settings and JVM used, this functionality could have no effect.

XML Editor Specific Actions

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

Split Actions

The editing area can be divided vertically and horizontally with the split / unsplit actions available on the **Split** toolbar, the **Document** > **Split** menu and the contextual menu of the editor panel for XML files:

- **Split Editor Horizontally**
- **Split Editor Vertically**
- **Unsplit Editor**

Edit Actions

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

- **Document** > **Edit** > **Toggle Line Wrap - (Ctrl + Shift + Y)** Turns on line wrapping in the editor panel if it was off and vice versa. It has the same effect as the *Line wrap* preference.
- **Document** > **Edit** > **Toggle comment (Ctrl + Shift + ,)** - Comments the current selection of the current editor. If the selection already contains a comment the action removes the comment from around the selection. If there is no selection in the current editor and the cursor is not positioned inside a comment the current line is commented. If the cursor is positioned inside a comment then the commented text is uncommented. The action is also available on the popup menu of the editor panel.

Select Actions

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

- **Document** > **Select** > **Element** - Selects the entire current element.
- **Document** > **Select** > **Content** - Selects the content of the current element, excluding the start tag and end tag. If it is applied repeatedly, starts with selecting the XML element from the cursor position and extends the selection to the ancestor XML elements. Each execution of the action extends the current selection to the surrounding element.
- **Document** > **Select** > **Attributes** - Selects all the attributes of the current element.
- **Document** > **Select** > **Parent** - Selects the parent element of the current element.
- Triple click on an element or processing instruction - If the triple click is done before the start tag of an element or after the end tag of an element then all the element is selected by the triple click action. If it is done after the start tag or before the end tag then only the element content without the start tag and end tag is selected.
- 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:

- **Document** > **Source** > **Locks / Unlocks the XML Tags** - Disables / Enables editing of XML tags.
- **Document** > **Source** > **To lower case** - Converts the selection's content to lower case characters.
- **Document** > **Source** > **To upper case** - Converts the selection's content to upper case characters.
- **Document** > **Source** > **Capitalize lines** - Converts to upper case the first character of every selected line.
- **Document** > **Source** > **Shift Right (Tab)** - Shifts the currently selected block to the right.
- **(Shift+Tab)** > **Document** > **Source** > **Shift Left (Shift+Tab)** - Shifts the selected block to the left.
- **Document** > **Source** > **Escape Selection ...** - Escapes a range of characters by replacing them with the corresponding character entities.
- **Document** > **Source** > **Unescape Selection ...** - Replaces the character entities with the corresponding characters.
- **Document** > **Source** > **Indent selection (Ctrl + I)** - Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences of the user*.

- **Document** > **Source** > **Format and Indent Element (Ctrl + Shift + I)** - Pretty prints the element that surrounds the caret position.
- **Document** > **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">
```

- **Document** > **Source** > **To Lower Case** - The action works on the selection converting all upper case letters to lower case.
- **Document** > **Source** > **To Upper Case** - The action works on the selection converting all lower case letters to upper case.
- **Document** > **Source** > **Capitalize lines** - It capitalizes the first letter found on every new line that is selected. Only the first letter is affected, the rest of the line remains the same. If the first character on the new line is not a letter then no changes are made.
- **Document** > **Source** > **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.
- **Document** > **Source** > **Insert new line after** : This useful action has the same result with moving the caret to the end of the current line and pressing *ENTER*.

XML Document Actions

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

- **Document** > **Schema** > **Show Definition** (also available on the contextual menu of the editor panel) - Moves the cursor to the definition of the current element in the schema associated with the edited XML document (DTD, XML Schema, Relax NG schema).
- **Document** > **XML Document** > **Copy XPath (Ctrl+Alt+.)** - Copies the XPath expression of the current element or attribute from the current editor to the clipboard.
- **Document** > **XML Document** > **Go to Matching Tag (Ctrl+Shift+G)** - Moves the cursor to the end tag that matches the start tag, or vice versa.
- **Document** > **XML Document** > **Go after Next Tag (Ctrl+])** - Moves the cursor to the end of the next tag.
- **Document** > **XML Document** > **Go after Previous Tag (Ctrl+[)** - Moves the cursor to the end of the previous tag.
- **Document** > **XML Document** > **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 Author fully implements the W3C recommendation regarding *Associating Style Sheets with XML documents*. See also *Specifying external style sheets* in HTML documents.

XML Refactoring Actions

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

- **Document** > **XML Refactoring** > **Surround with tag... (Ctrl+E)** - Selected text in the editor is marked with the specified start and end tags.

- **Document** > **XML Refactoring** > **Surround with last <tag> (Ctrl+/)** - Selected Text in the editor is marked with start and end tags of the last 'Surround in' action.
- **Document** > **XML Refactoring** > **Rename element (Alt+Shift+R)** - 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.

  Document > **XML Refactoring** > **Rename prefix (Alt+Shift+P)** - 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 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.
- **Document** > **XML Refactoring** > **Split element (Ctrl+Alt+D)** - Split the element from the caret position in two identical elements. The caret must be inside the element.
- **Document** > **XML Refactoring** > **Join elements (Ctrl+Alt+J)** - Joins the left and right elements relative to the current caret position. The elements must have the same name, attributes and attributes values.
- **Document** > **XML Refactoring** > **Delete element tags (Ctrl+Alt+X)** - Deletes the start and end tag of the current element.

Smart Editing

The following helper actions are available in the XML editor:

- *Closing tag auto-expansion* - If you want to insert content into an auto closing tag like <tag/> deleting the / character saves some keystrokes by inserting a separate closing tag automatically and placing the cursor between the start and end tags: <tag></tag>
- *Auto-rename matching tag* - When you edit the name of the start tag, Oxygen 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.
- *Triple click* - A triple click with the left mouse button 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 triple 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.

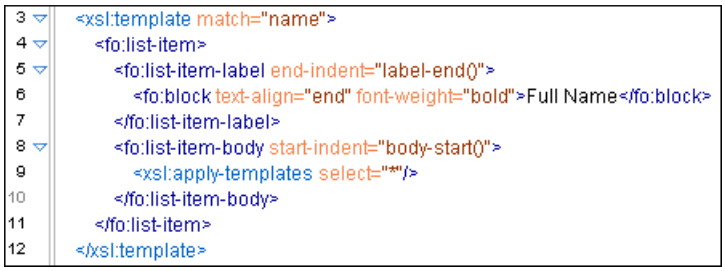


Figure 46: Example of Coloring XML Tags by Prefix

Editing XML Schemas

An XML Schema describes the structure of an XML document and is used to validate XML document instances against it, in order 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.

Oxygen provides two editing pages for working with XML Schema: the usual *Text* editing page and the visual *Design* editing page.

XML Schema Text Editor

This page 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. To activate a side by side source and diagram presentation you have to enable the *Show XML Schema Diagram* checkbox 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.

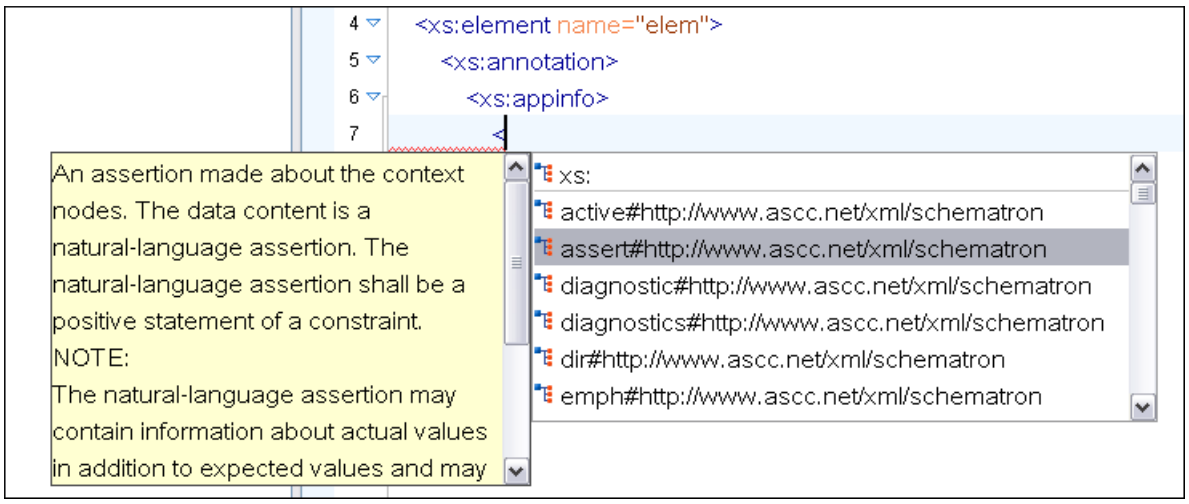


Figure 47: Schematron Support in XML Schema Content Completion

References to XML Schema Specification

The same as in editing XML documents, the message of an error obtained by validation of an XML Schema document includes a reference to the W3C specification for XML Schema. An error message contains an *Info* field that will open the browser on the "XML Schema Part 1:Structures" specification at exactly the point where the error is described thus allowing you to understand the reason for that error.

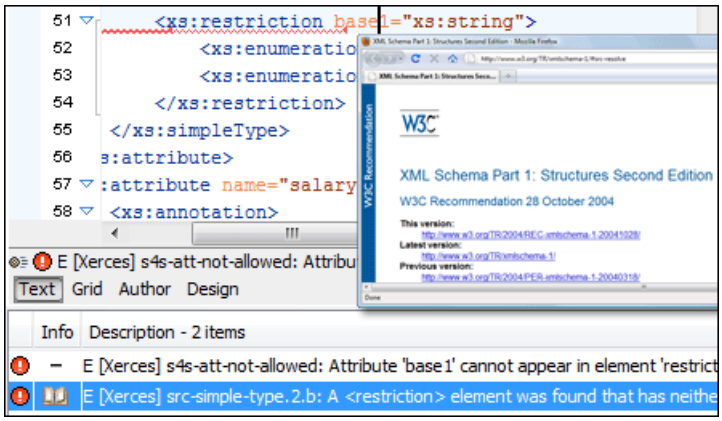


Figure 48: Link to Specification for XML Schema Errors

Validation of an XML Schema containing a type definition with a `minOccurs` or `maxOccurs` attribute having a value larger than 256 limits the value to 256 and issues a warning about this restriction in the Message panel at the bottom of the Oxygen window. Otherwise, for large values of the `minOccurs` and `maxOccurs` attributes the validator fails with an OutOfMemory error which practically makes Oxygen unusable without a restart of the entire application.

Important:

If the schema imports only a namespace without specifying the schema location and a *catalog is set-up* mapping the namespace to a certain location both validation and the schema components outline will correctly identify the imported schema.

XML Schema Actions

- The **Show Definition** action accessed from the **menu Document** > **Schema** > **Show Definition (Ctrl + Shift + ENTER)** moves the cursor to the definition of the referenced XML Schema item. The referenced item can be an element, group, simple type or complex type. The same action is executed on a double click on a component name

in the *Schema Outline view*. You can define a scope for this action in the same manner you define it for *Search Declarations*.

Flatten an XML Schema

If an XML Schema is organized on several levels linked by `xs:include` statements, sometimes it is more convenient to work on the schema as a single flat file. To flatten schema, Oxygen recursively adds included files to the master one. That means Oxygen replaces the `xs:include` elements with the ones coming from the included files.

This action works at file level not at schema document level so it is available only in Text mode of XML Schema editor. It can be accessed from the XML Schema text editor's **contextual menu** > **Refactoring** > **Flatten Schema** . Alternatively you can select one or more schemas in the **Project** view and invoke the action from the view's contextual menu. In this last case the feedback of the action will be presented in the **Information** view.

Schema flattening can also be accessed from command line by running scripts that come with Oxygen installation:

- `flattenSchema.bat` on Windows;
- `flattenSchema.sh` on Mac OS X and Unix/Linux.

The input file is the first argument of the script and the output file is the second argument.

In the following example `master.xsd` includes `slave.xsd`. This, in turn, includes `slave1.xsd` which includes both `slave2.xsd` and `slave3.xsd`.

`master.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="tns" xmlns:tns="tns"
    xmlns:b="b" >
  <!-- included elements from slave.xsd -->
  <xs:include schemaLocation="slave.xsd"></xs:include>
  <!-- master.xsd -->
  <xs:element name="element1">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:element2" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

`slave.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="tns" xmlns:a="a" xmlns:b="b"
    xmlns:c="c">
  <!-- included elements from slave1.xsd -->
  <xs:include schemaLocation="slave1.xsd"></xs:include>
  <!-- slave  -->
  <xs:element name="element2" xmlns:c="x"/>
</xs:schema>
```

`slave1.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="tns" xmlns:tns="tns"
    blockDefault=" restriction">
  <!-- included elements from slave2.xsd -->
  <xs:include schemaLocation="slave2.xsd"></xs:include>
  <!-- included elements from slave3.xsd -->
  <xs:include schemaLocation="slave3.xsd"></xs:include>
```

```
  <!-- slave1  -->
  <xs:element name="element0"/>
  <xs:element name="element7"/>
  <xs:element name="element7Substitute"
      substitutionGroup="tns:element7"
      block="extension"/>
  <xs:element name="element6">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:element7"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="type1">
    <xs:sequence>
      <xs:element ref="tns:element0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

slave2.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="tns"
  xmlns:tns="tns"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">
  <!-- slave2 -->
  <xs:element name="a"></xs:element>
  <a:element name="element9"
        xmlns:a="http://www.w3.org/2001/XMLSchema">
    <xs:complexType>
      <xs:sequence>
        <!-- This element is from the target namespace -->
        <xs:element name="element3"
              xmlns:b="http://www.w3.org/2001/XMLSchema"/>
        <!-- Element from no namespace -->
        <xs:element name="element4" form="unqualified"/>
        <a:element ref="tns:a"></a:element>
      </xs:sequence>
      <!-- Attribute from the target namespace -->
      <b:attribute name="attr1" type="xs:string"
            xmlns:b="http://www.w3.org/2001/XMLSchema"/>
      <!-- Attribute from the no namespace -->
      <xs:attribute name="attr2" type="xs:string"
            form="unqualified"/>
    </xs:complexType>
  </a:element>
</xs:schema>
```

slave3.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="tns" finalDefault="restriction"
    xmlns:tns="tns">
  <!-- slave3 -->
  <xs:complexType name="ct1"/>
  <xs:complexType name="ct2" final="extension">
    <xs:complexContent>
      <xs:extension base="tns:ct1"/>
    </xs:complexContent>
  </xs:complexType>
```

```
  <xs:simpleType name="st1" final="union">
    <xs:restriction base="xs:integer"/>
  </xs:simpleType>
  <xs:simpleType name="st2" final="union">
    <xs:restriction base="tns:st1">
      <xs:enumeration value="1"/>
      <xs:enumeration value="2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="e1" type="tns:c1" final="restriction"/>
  <xs:element name="e2ext" type="tns:c2"
        substitutionGroup="tns:e1"></xs:element>
  <xs:complexType name="c1">
    <xs:sequence>
      <xs:element ref="tns:e1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="c2">
    <xs:complexContent>
      <xs:extension base="tns:c1">
        <xs:sequence>
          <xs:element ref="tns:e1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

`master.xsd` after it has been flattened:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="tns" xmlns:a="a"
      xmlns:b="b" xmlns:c="c" xmlns:tns="tns"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- included elements from slave.xsd -->
  <!-- included elements from slave1.xsd -->
  <!-- included elements from slave2.xsd -->
  <!-- slave2 -->
  <xs:element block="restriction" name="a"/>
  <a:element block="restriction" name="element9"
        xmlns:a="http://www.w3.org/2001/XMLSchema">
    <xs:complexType>
      <xs:sequence>
        <!-- This element is from the target namespace -->
        <xs:element block="restriction" form="qualified"
name="element3"
          xmlns:b="http://www.w3.org/2001/XMLSchema"/>
        <!-- Element from no namespace -->
        <xs:element block="restriction" form="unqualified"
              name="element4"/>
        <a:element ref="tns:a"/>
      </xs:sequence>
      <!-- Attribute from the target namespace -->
      <b:attribute form="qualified" name="attr1" type="xs:string"
        xmlns:b="http://www.w3.org/2001/XMLSchema"/>
      <!-- Attribute from the no namespace -->
      <xs:attribute form="unqualified" name="attr2"
type="xs:string"/>
    </xs:complexType>
  </a:element>
  <!-- included elements from slave3.xsd -->
  <!-- slave3 -->
  <xs:complexType block="restriction" final="restriction"
```

```
name="ct1"/>
  <xs:complexType block="restriction" final="extension" name="ct2">

    <xs:complexContent>
      <xs:extension base="tns:ct1"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType final="union" name="st1">
    <xs:restriction base="xs:integer"/>
  </xs:simpleType>
  <xs:simpleType final="union" name="st2">
    <xs:restriction base="tns:st1">
      <xs:enumeration value="1"/>
      <xs:enumeration value="2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element block="restriction" final="restriction" name="e1"
        type="tns:c1"/>
  <xs:element block="restriction" final="restriction" name="e2ext"

        substitutionGroup="tns:e1"
    type="tns:c2"/>
  <xs:complexType block="restriction" final="restriction"
        name="c1">
    <xs:sequence>
      <xs:element ref="tns:e1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType block="restriction" final="restriction"
        name="c2">
    <xs:complexContent>
      <xs:extension base="tns:c1">
        <xs:sequence>
          <xs:element ref="tns:e1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- slave1  -->
  <xs:element block="restriction" name="element0"/>
  <xs:element block="restriction" name="element7"/>
  <xs:element block="extension" name="element7Substitute"
        substitutionGroup="tns:element7"/>
  <xs:element block="restriction" name="element6">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:element7"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType block="restriction" name="type1">
    <xs:sequence>
      <xs:element ref="tns:element0"/>
    </xs:sequence>
  </xs:complexType>
  <!-- slave  -->
  <xs:element name="element2" xmlns:c="x"/>
  <!-- master.xsd -->
  <xs:element name="element1">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:element2"/>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The case of XML Schema redefinitions is also handled as shown in the example below.

`master.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:redefine schemaLocation="slave1.xsd">
    <xs:complexType name="tp">
      <xs:complexContent>
        <xs:extension base="tp">
          <xs:choice>
            <xs:element name="el2" type="xs:NCName"/>
            <xs:element name="el3" type="xs:string"/>
          </xs:choice>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:redefine>
  <xs:element name="el" type="tp"/>
</xs:schema>
```

`slave1.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:redefine schemaLocation="slave2.xsd">
    <xs:complexType name="tp">
      <xs:complexContent>
        <xs:extension base="tp">
          <xs:attribute name="a"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:redefine>
</xs:schema>
```

`slave2.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="tp">
    <xs:sequence>
      <xs:element name="el" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

`master.xsd` after it has been flattened:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="tp">
    <xs:complexContent>
      <xs:extension base="tp_Redefined1">
        <xs:choice>
          <xs:element name="el2" type="xs:NCName"/>
          <xs:element name="el3" type="xs:string"/>
        </xs:choice>
      </xs:extension>
```

```
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="tp_Redefined1">
    <xs:complexContent>
      <xs:extension base="tp_Redefined0">
        <xs:attribute name="a"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="tp_Redefined0">
    <xs:sequence>
      <xs:element name="el" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="el" type="tp"/>
</xs:schema>
```

The references to the included schema files can be resolved through an *XML Catalog*.

XML Schema Diagram Editor

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 provides a simple and expressive **Design** page for editing XML Schemas. The schema diagram helps both the content authors who want to understand a schema and schema designers who develop complex schemas.

The diagram font can be increased using the usual Oxygen shortcuts: **(Ctrl - +)** , **(Ctrl - -)** , **(Ctrl - 0)** or **(Ctrl - mouse wheel)**. The whole diagram can also be zoomed with one of the predefined factors *available in the Schema preferences panel*. The same zoom factor is applied for the print and save actions.

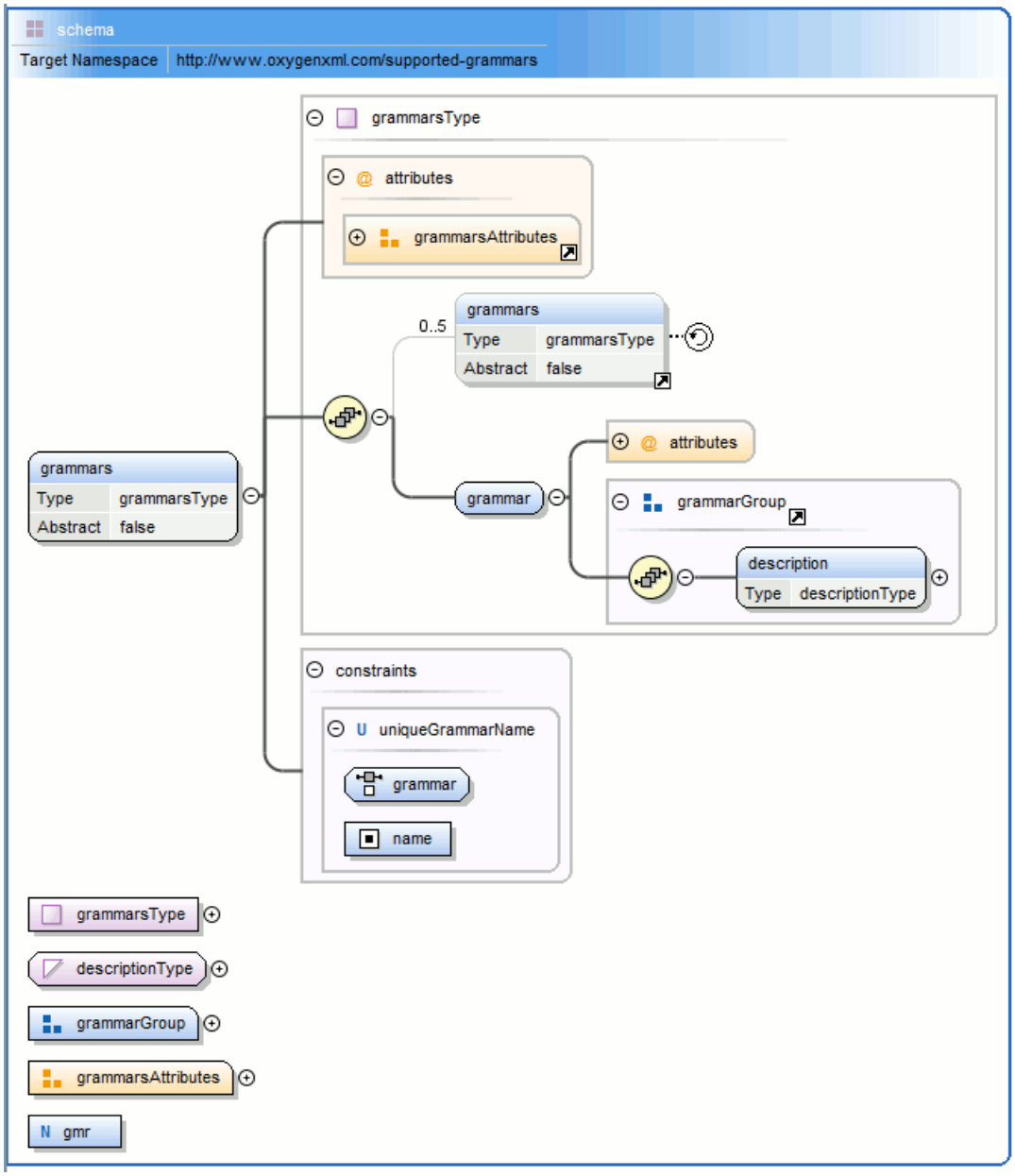
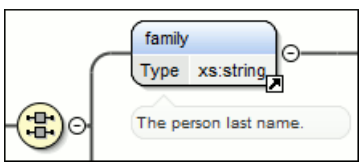


Figure 49: XML Schema Diagram

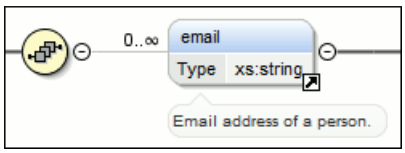
XML Schema Components

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);

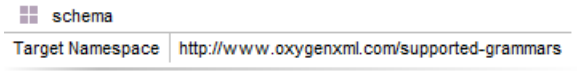


- a thin line to identify a connection with an optional component (in the following image, `email` is an optional element).



The following topics explain in detail all available components and their symbols as they appear in an XML schema diagram.

`xs:schema`



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/xmlschema-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 `block` attribute of `xs:element` and `xs:complexType`.	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty]
Final Default	Default value of the `final` attribute of `xs:element` and `xs:complexType`.	#all, restriction, extension, restriction extension, [Empty]
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/xmlschema-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.
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

Property Name	Description	Possible Values	Mentions
Substitution Group	Qualified name of the head of the substitution group to which this element belongs.	All declared elements	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
Nillable	When this attribute is set to true, the element can be declared as nil using an `xsi:nil` attribute in the instance documents.	true/false	For global elements and element references
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 `xsi:type` 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 `blockDefault` attribute of the parent `xs:schema`.	#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 `finalDefault` attribute of the parent `xs:schema`.	#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`

Defines an attribute. See more info at *http://www.w3.org/TR/xmlschema-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.
Use	Possible usage of the attribute. Marking an attribute "prohibited" is useful to exclude attributes	optional, required, prohibited	For local attributes

Property Name	Description	Possible Value	Mentions
	during derivations by restriction.		
Form	Specifies if the attribute is qualified (i.e., must have a namespace prefix in the instance XML document) or not. The default value for this attribute is specified by the `attributeFormDefault` attribute of the `xs:schema` document element.	unqualified/qualified	For local attributes.
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:complexType`

Defines a top level complex type. Complex Type Definitions provide for: See more data at *http://www.w3.org/TR/xmlschema-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.

Property Name	Description	Possible Values	Mentions
Content	The content of the complex type.	simple/ complex	For complex types which extend/restrict a base type. It is automatically detected.
Content Mixed	Specifies if the complex content model will be mixed.	true/false	For complex contents.
Mixed	Specifies if the complex type content model will be mixed.	true/false	For global and anonymous complex types.
Abstract	When set to `true`, this complex type cannot be used directly in the instance documents and needs to be substituted using an `xsi:type` attribute.	true/false	For global and anonymous complex types.
Block	Controls whether a substitution (either through a `xsi:type` or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the `blockDefault` attribute of `xs:schema`.	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.
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`

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/xmlschema-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:group`

Defines a group of elements to be used in complex type definitions. See more info at *http://www.w3.org/TR/xmlschema-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:attributeGroup`

Defines an attribute group to be used in complex type definitions. See more info at *http://www.w3.org/TR/xmlschema-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.

Property Name	Description	Possible Values	Mentions
ID	The component id.	Any id	For all attribute groups.
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:include`

Adds multiple schemas with the same target namespace to a document. See more info at *http://www.w3.org/TR/xmlschema-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/xmlschema-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
Component	The component name	Not editable property.

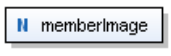
xs:redefine

Redefines simple and complex types, groups, and attribute groups from an external schema. See more info at *http://www.w3.org/TR/xmlschema-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:notation`



Describes the format of non-XML data within an XML document. See more info at *http://www.w3.org/TR/xmlschema-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 50: 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/xmlschema-1/#element-sequence*.

Figure 51: 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/xmlschema-1/#element-choice*.

Figure 52: An `xs:all` in diagram

`xs:all` specifies that the child elements can appear in any order. Each child element can occur 0 or 1 time. See more info at *http://www.w3.org/TR/xmlschema-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.

Property Name	Description	Possible Values	Mentions
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`

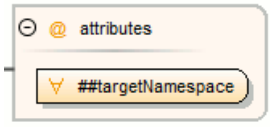
Enables the author to extend the XML document with elements not specified by the schema. See more info at *http://www.w3.org/TR/xmlschema-1/#element-any*.

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
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`

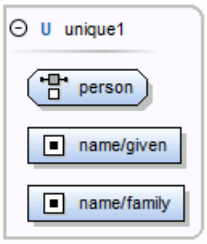


Enables the author to extend the XML document with attributes not specified by the schema. See more info at *http://www.w3.org/TR/xmlschema-1/#element-anyAttribute*.

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`

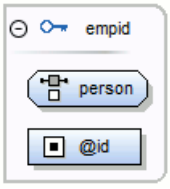


Defines that an element or an attribute value must be unique within the scope. See more info at *http://www.w3.org/TR/xmlschema-1/#element-unique*.

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`



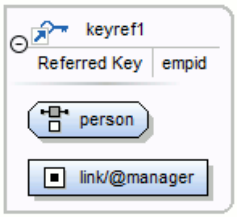
Specifies an attribute or element value as a key (unique, non-nullable and always present) within the containing element in an instance document. See more info at *http://www.w3.org/TR/xmlschema-1/#element-key*.

xs:key properties

Property Name	Description	Possible Value
Name	The key name. Always required.	Any NCName.

Property Name	Description	Possible Value
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`



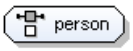
Specifies that an attribute or element value corresponds to that of the specified key or unique element. See more info at *http://www.w3.org/TR/xmlschema-1/#element-keyref*.

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/xmlschema-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/xmlschema-1/#element-field*.

xs:field properties

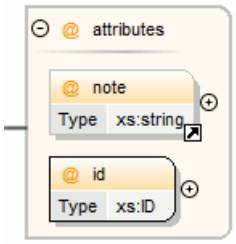
Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the field(s) composing the key, key reference, or unique constraint.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not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

Constructs Used to Group Schema Components

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

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

Attributes

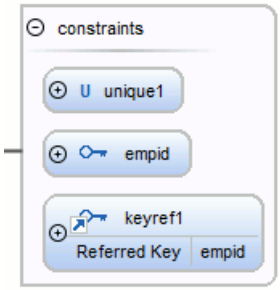


Groups all attributes and attribute groups belonging to a complex type.

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

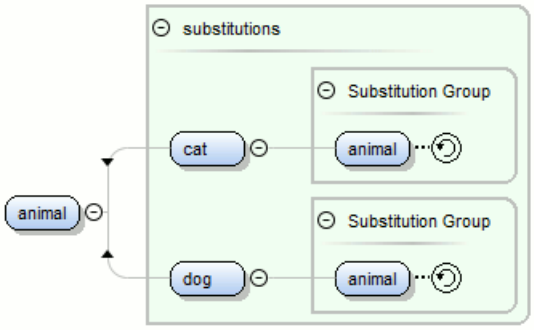


Groups all constraints (*xs:key*, *xs:keyRef* or *xs:unique*) belonging to an element.

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.

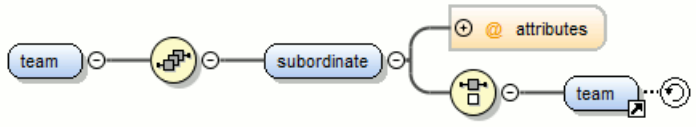
Navigation in the Schema Diagram

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 to . You can also make discontinuous selections in the schema diagram using the *Ctrl* key.
- Use the arrow keys to navigate the diagram vertically and horizontally.
- Use *Home*/*End* keys to navigate to the first/last component from the same level. Use **(Ctrl - Home)** key combination to go to the diagram root and **(Ctrl - 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.
- Search in the diagram using the *Find/Replace dialog* or the *Quick find toolbar*. You can find/replace components only in the current file scope.
- You can expand and see the contents of the imports/includes/redefines in the diagram. In order to edit components from other schemas the schema for each component will be opened as a separate file in Oxygen.

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** page is seamlessly integrated in the Oxygen *XML documents validation* capability.

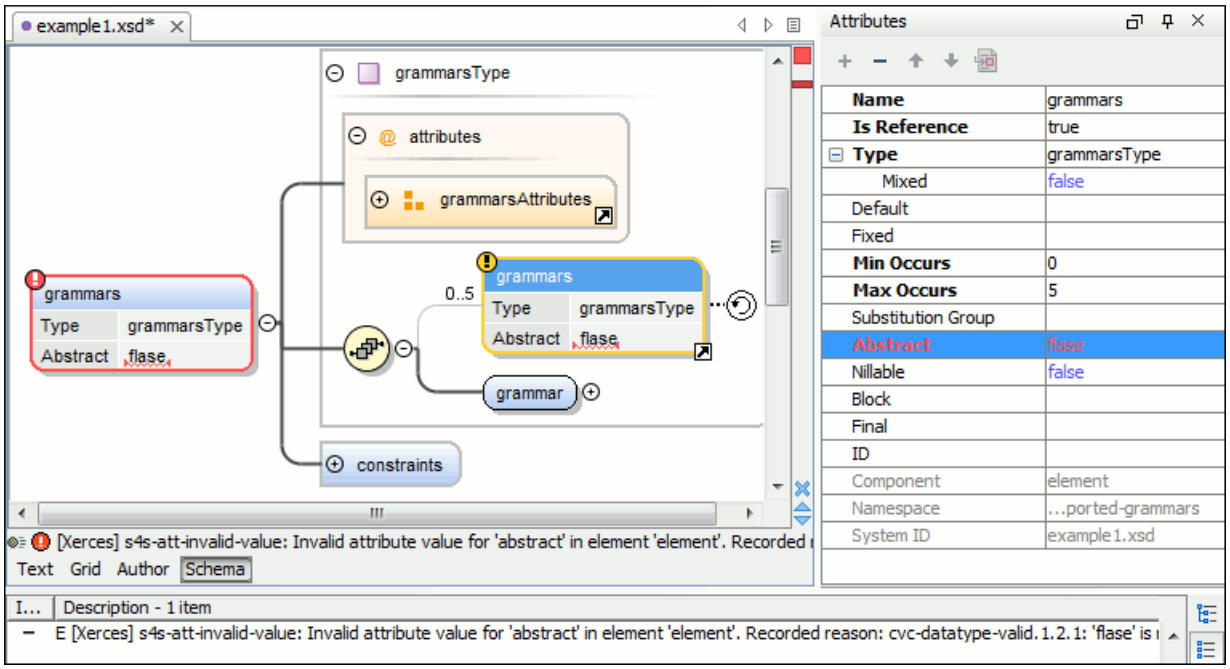


Figure 53: XML Schema Validation

A schema validation error is presented by highlighting the invalid component in the following places:

- in the *Attributes View*
- in the diagram by surrounding the component that has the error with a red border
- a marker on the errors stripe at the right of the diagram view
- a status label with a red icon () below the diagram view

Invalid facets for a component are highlighted in the *Facets View*.

Components with invalid properties are rendered with a red border. This is a default color, but you can customize it in the *Document checking user preferences*. When hovering an invalid component, the tooltip will present the validation errors associated with that component.

When editing a value which is supposed to be a qualified or unqualified XML name, the application provides automatic validation of the entered value. This proves to be very useful in avoiding setting invalid XML names for the given property.

If you validate the entire schema using **Document** > **Validate Document (Ctrl+Shift+V)** 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.

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

The schema can be edited using drag and drop operations or contextual menu actions.

Drag and drop action provides the easiest way to move the existing components to other locations in the schema. For example, an element reference can be quickly inserted 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)** 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/redefine component. An edit dialog is displayed, allowing you to customize the directives.

The contextual menu of the **Design** page offers the following edit actions:

- **Show Definition** (**(Ctrl - 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 - 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...** () - 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.

  Note: For imported/included components which do not belong to the current edited schema the dialog presents the annotation as read-only and you will have to open the schema where the component is defined in order to edit its annotation.

  Note: 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.

- **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.

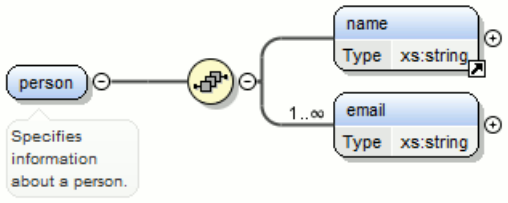
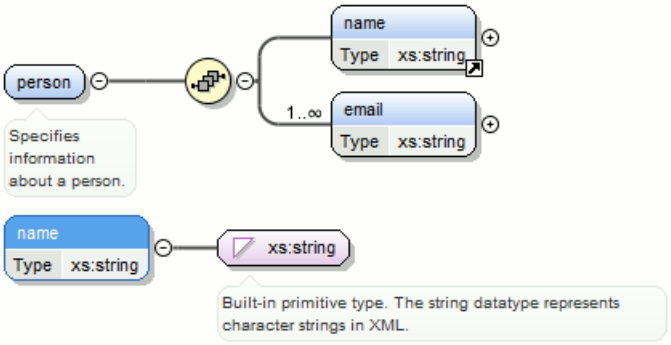


Figure 54: Extracting a Global Element

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.

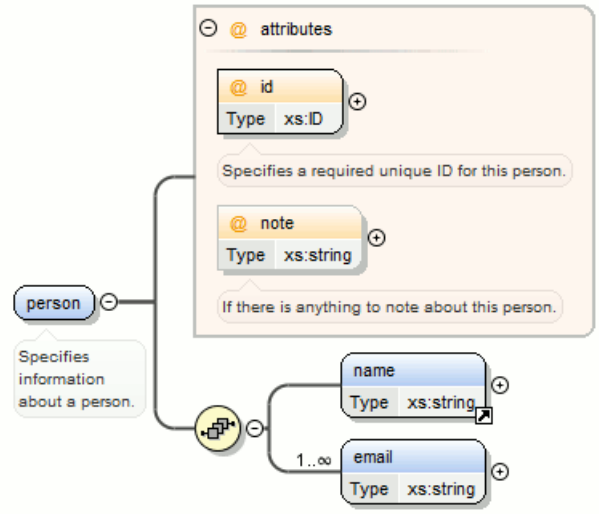
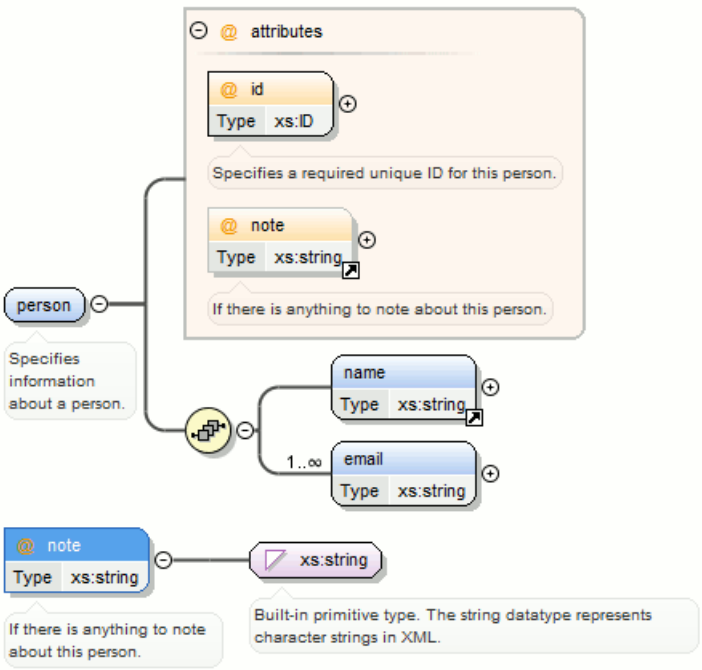
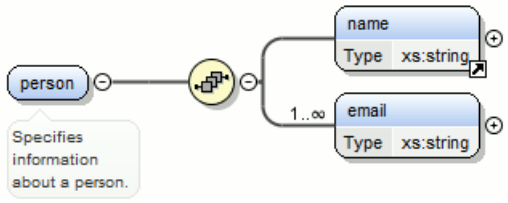


Figure 55: Extracting a Global Attribute

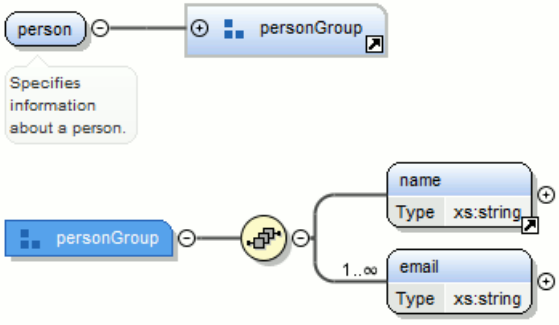
If you execute **Extract Global Attribute** on attribute `note` the result is:



- **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 56: 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.

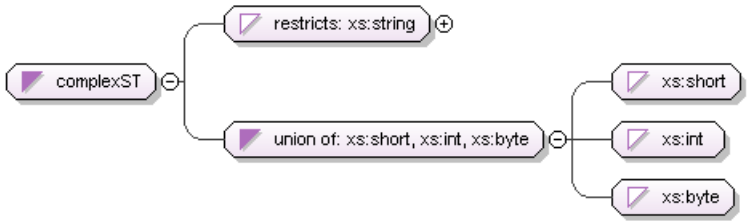


Figure 57: Extracting a Global Simple Type

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

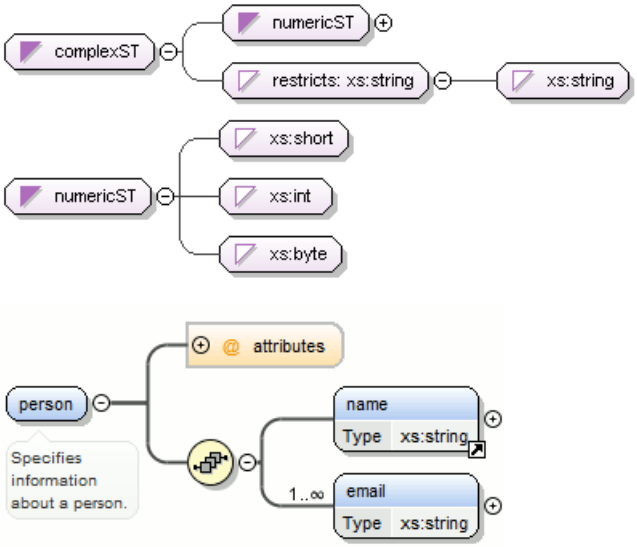
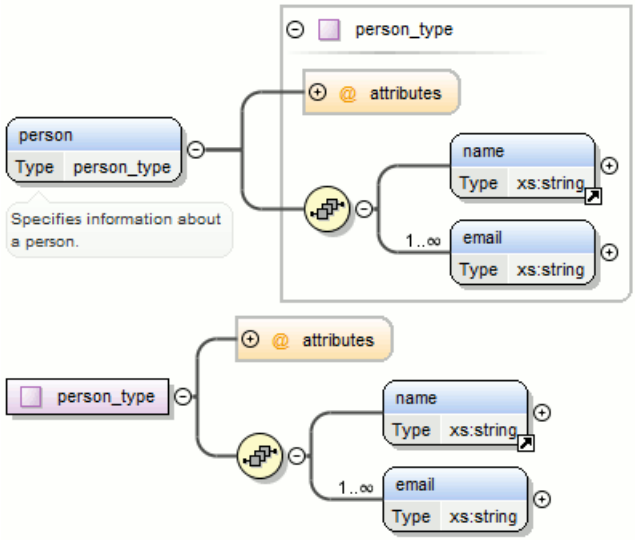


Figure 58: Extracting a Global Complex Type

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 - X)** - Cut the selected component(s).
- **Copy (Ctrl - C)** - Copy the selected component(s).

-  **Paste (Ctrl - 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).
- **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.
- **Resource Dependencies** - Allows you to see the dependencies for the current selected resource.
-  **Expand all** - Expands recursively all sub-components of the selected component.
-  **Collapse all** - Collapses recursively all sub-components of the selected component.
- **Save as Image...** - Save the diagram as image, in JPEG, BMP, SVG or PNG format.
-  **Generate Sample XML Files** - Generate XML files using the current opened schema. The selected component is the XML document root. See more in the *Generate Sample XML Files* section.
-  **Options...** - Show the *Schema preferences panel*.

Schema Outline View

The **Outline** view presents all the global components grouped by their location, namespace, or type. If hidden, you can open it from **Window** > **Show View** > **Outline** .

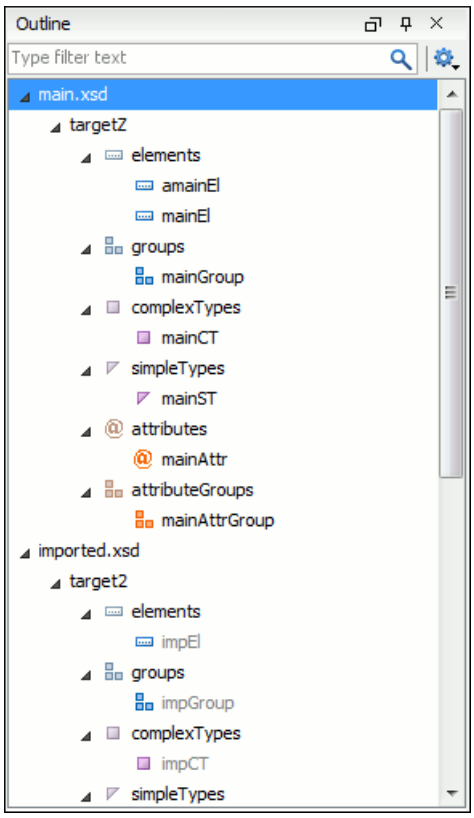


Figure 59: The Outline View for XML Schema

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, you need to 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 local components** - Displays components defined in the current file only.
- **Group by location/namespace/type** - These three operations allow you to group the components by location, namespace, or type. When grouping by namespace, the main schema target namespace is the first presented in the **Outline** view.

The following contextual menu actions are available:

- **Remove (Delete)** - Removes the selected item from the diagram.
- **Search References (Ctrl-Shift-R)** - Searches all references of the item found at current cursor position in the defined scope, if any.
- **Search References in...** - Searches all references of the item found at current cursor position in the specified scope.
- **Component Dependencies (Ctrl + Shift + F4)** - Allows you to see the dependencies for the current selected component.
- **Resource Hierarchy (F4)** - Allows you to see the hierarchy for the current selected resource.
- **Resource Dependencies (Shift + F4)** - Allows you to see the dependencies for the current selected resource.
- **Rename Component** - 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 is synchronized with **Text** view; when you click a component in the **Outline** view, its definition is highlighted in the **Text** view.

The Attributes View

The **Attributes** view presents the properties for the selected component in the schema diagram. If hidden, you can open it from **Window** > **Show View** > **Attributes** .

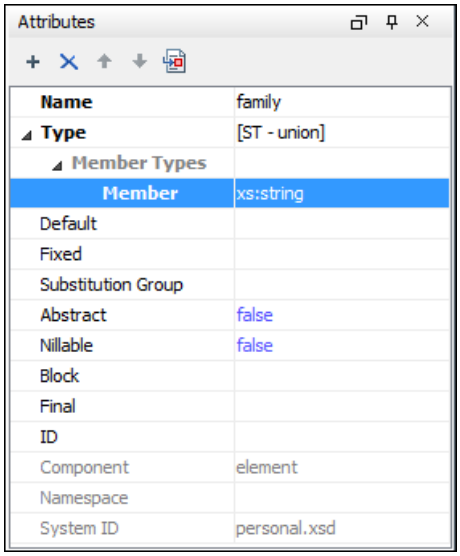


Figure 60: The Attributes View

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** - Show the definition for the selected type.
- **Show Facets** - Allows you to edit the facets for a simple type.

The Facets View

The **Facets** view presents the facets for the selected component, if available. If hidden, you can open it from **Window** > **Show View** > **Facets** .

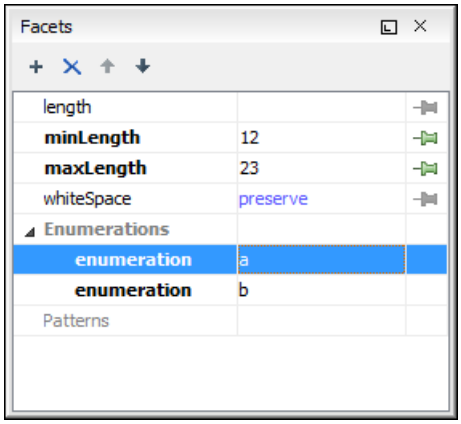


Figure 61: The Facets View

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 can edit regular expressions either be hand or you can right click, choose **Open in Regular Expression Builder** and have a full-fledged *XML Schema Regular Expression builder* to guide you in testing and constructing the pattern.

The Palette View

Designed to offer quick access to XML Schema components, the **Palette** view improves the usability of the XML Schema diagram builder allowing you drag components from the **Palette** view and drop them into the **Design** page.

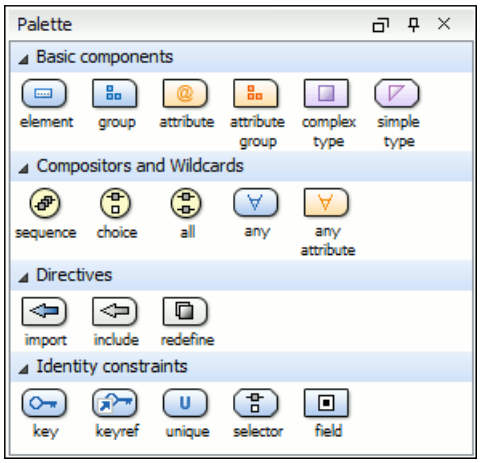


Figure 62: Palette View

Components are organized functionally into 4 collapsible categories:

- Basic components: *elements*, *group*, *attribute*, *attribute group*, *complex type*, *simple type*.
- Compositors and Wildcards: *sequence*, *choice*, *all*, *any*, *any attribute*.
- Directives: *import*, *include*, *redefine*.
- Identity constraints: *key*, *keyref*, *unique*, *selector*, *field*.

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).

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** page 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.

Contextual Editing

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, a function defined in a main schema document is not visible when you edit an included or imported module. Oxygen 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.

To set a main schema files, you need to define a validation scenario and add validation units that point to the main schemas. Oxygen 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 include components defined in modules other than the currently edited one;
- the **Outline** displays the components collected from the entire schema structure;

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

To generate sample XML files from an XML Schema use the **Tools** > **Generate Sample XML Files...** action. The action is also available on the contextual menu from the schema *Design page*.

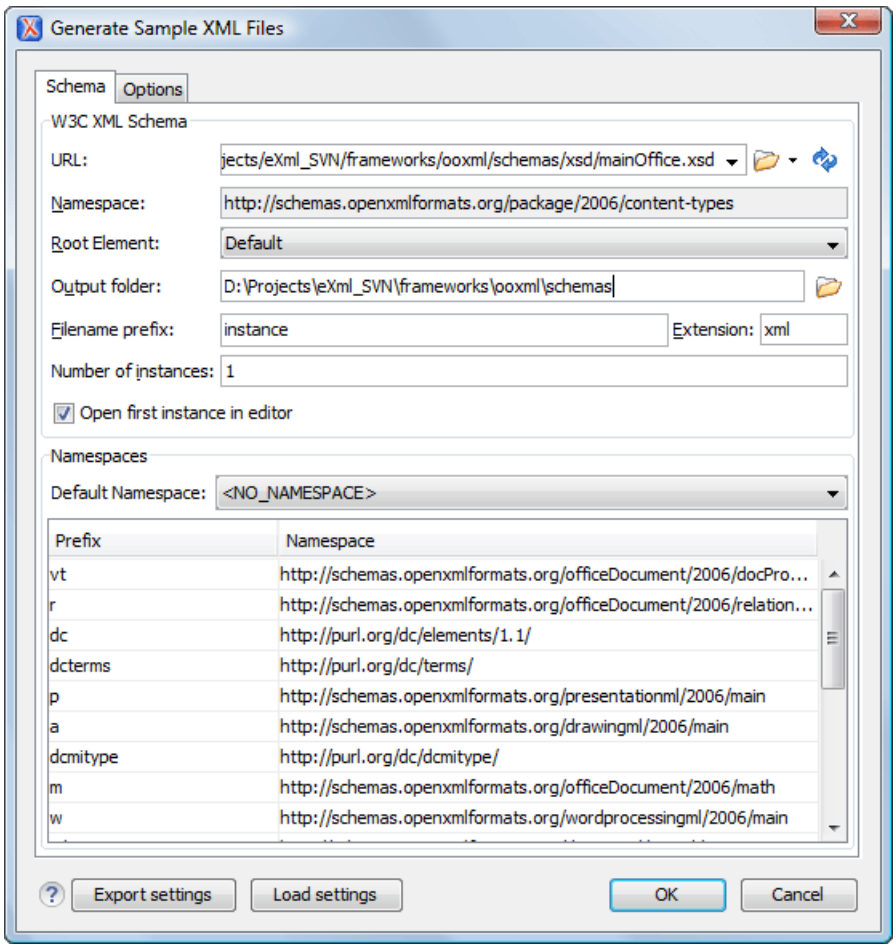


Figure 63: The Generate Sample XML Files Dialog

Complete the dialog as follows:

- **URL** - Schema's URL. Last used URLs are displayed in the drop-down box.
- **Namespace** - Displays the namespace of the selected schema.
- **Document root** - After the list is selected, a list of elements is displayed in the combo box. The user should choose the root of the XML documents to be generated.
- **Output folder** - Path to the folder where the generated XML instances will be saved.
- **Filename prefix and Extension** - Generated files' names have the following format: `prefixN.extension`, where `prefix` and `extension` are specified by the user and `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 will be opened in editor.
- **Namespaces** - Here the user 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 becomes active only after the URL field is filled-in and a schema is detected. It allows the user to set specific options for different namespaces and elements.

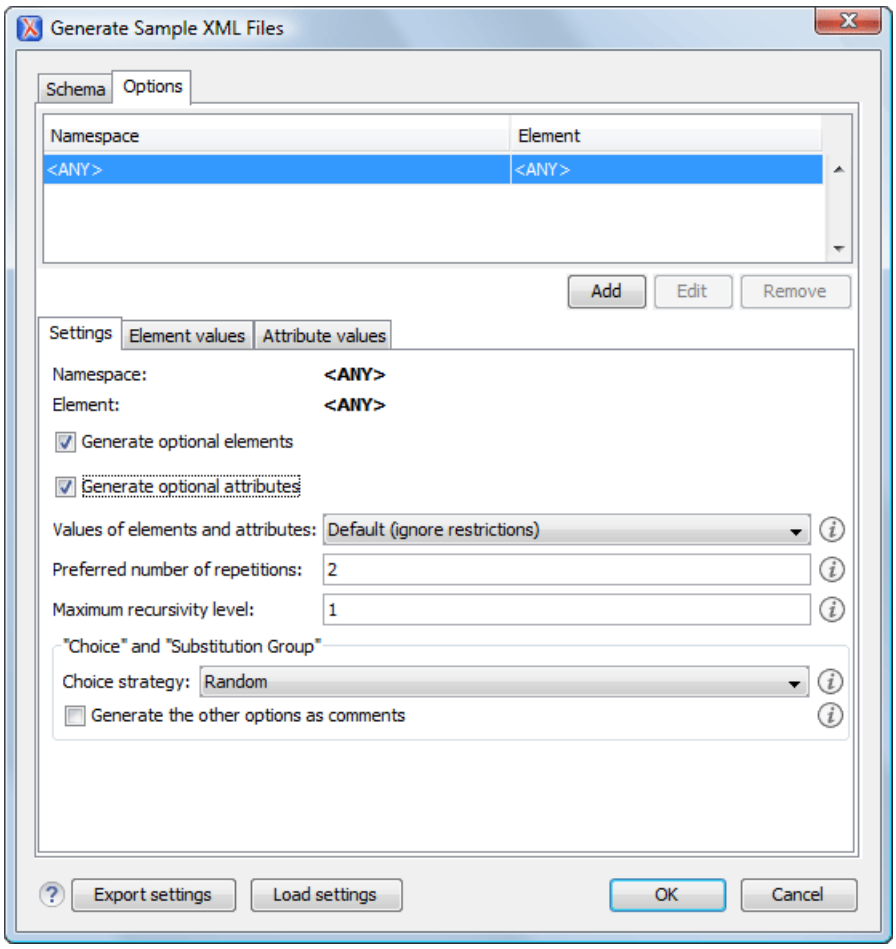


Figure 64: The Generate Sample XML Files Dialog

The **Namespace / Element** table allows you to set a namespace for each element name that will appear in an XML document instance.

- **Namespace / Element table** -
  - All elements from all namespaces. This is the default setting and it can also be *accessed* from **Options** > **Preferences** > **XML** > **XML Instance Generator** .
  - All elements from a specific namespace.
  - A specific element from a specific namespace.
- **Settings**
  - **Generate optional elements** - When checked, all elements will be generated, including the optional ones (having the `minOccurs` attribute set to 0 in the schema).
  - **Generate optional attributes** - When checked, all attributes will be 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 attributes and elements. Several choices are available:
    - None - No content is inserted;
    - Default - Inserts a default value depending of data type descriptor of the respective 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). Please note that type restrictions are ignored for this option when generating the values of elements and attributes. 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. If you need to generate valid values, then the recommended option is **Random**.
    - Random - Inserts a random value depending of data type descriptor of the respective element or attribute.
- **Preferred number of repetitions** - Allows the user to set the preferred number of repeating elements related with `minOccurs` and `maxOccurs` defined in XML Schema.
    - If the value set here is between `minOccurs` and `maxOccurs`, then that value will be used;
    - If the value set here is less than `minOccurs`, then the `minOccurs` value will be used;
    - If the value set here is greater than `maxOccurs`, then that value will be used.
- **Maximum recursion level** - Option to set the maximum allowed depth of the same element if a recursion is found.
- **Choice strategy** - Option to be used in case of xs:choice or substitutionGroup. The possible strategies are:
    - **First** - the first branch of `xs:choice` or the head element of `substitutionGroup` will be always used;
    - **Random** - a random branch of `xs:choice` or a substitute element or the head element of a `substitutionGroup` will be used.
- **Generate the other options as comments** - Option to generate the other possible choices or substitutions (for `xs:choice` and `substitutionGroup`). These alternatives will be 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 will be used to generate the elements content. If there are more than one value, then the values will be used in a random order.

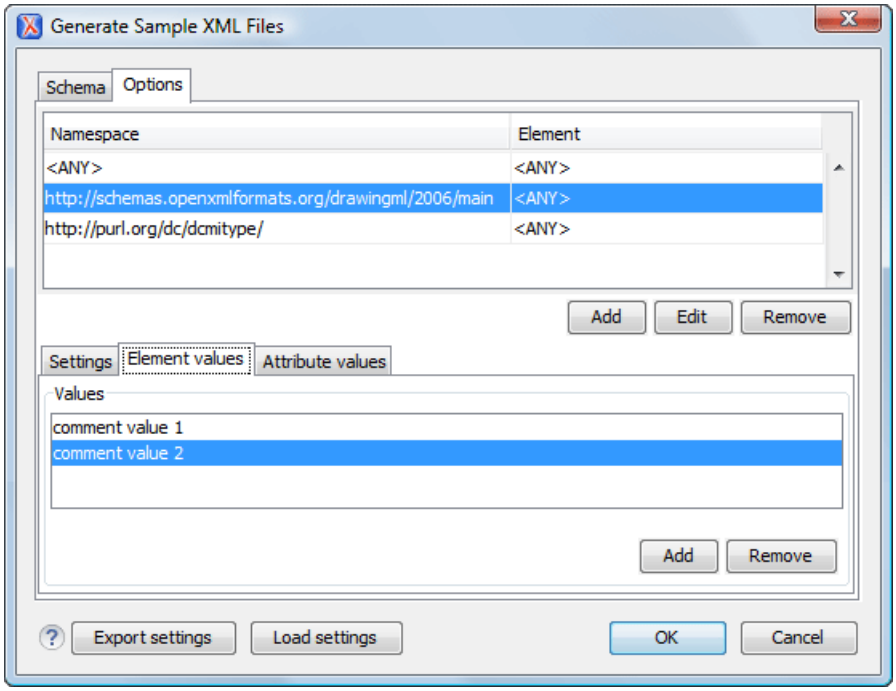


Figure 65: The Element Values Tab

- **Attribute values** - The **Attribute values** tab allows you to add values that will be used to generate the attributes content. If there are more than one value, then the values will be used in a random order.

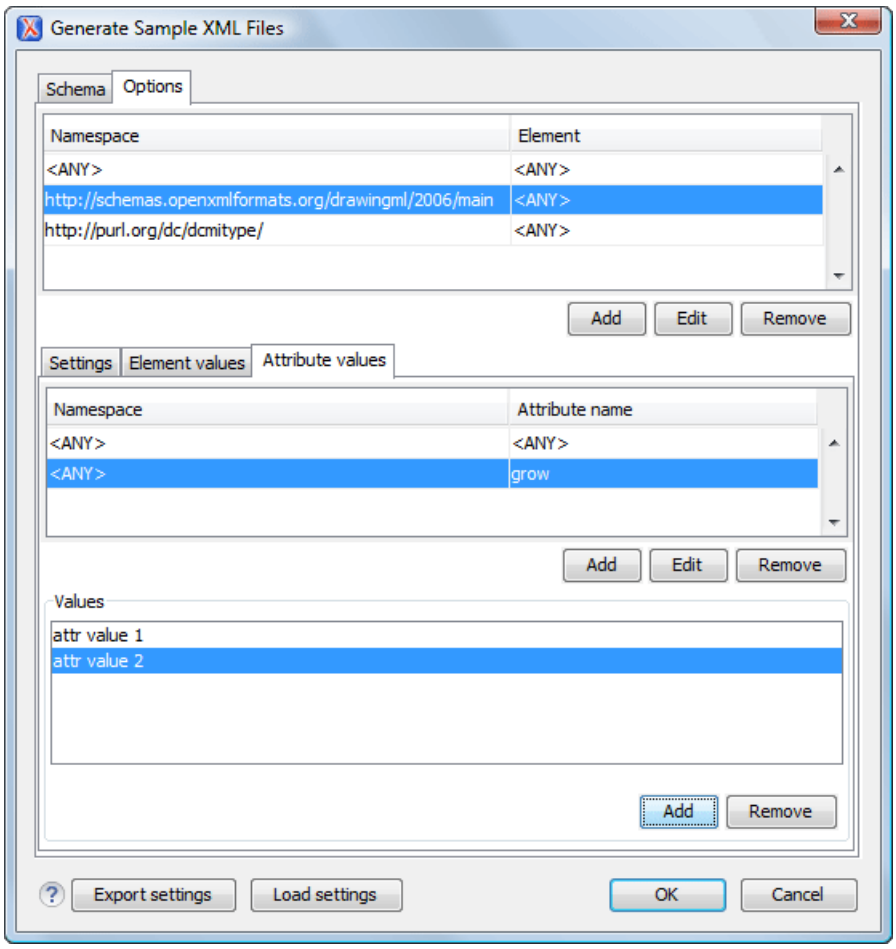


Figure 66: The Attribute Values Tab

Running the XML Instance Generator From Command Line

The XML instance generator tool can be also used from command line by running the script called `xmlGenerator.bat` (on Windows) / `xmlGenerator.sh` (on Mac OS X / Unix / Linux) located in the Oxygen installation folder. The parameters can be set once in the dialog, exported to an XML file on disk with the button **Export settings** and reused from command line. With the exported settings file you can generate the same XML instances from the command line as from the dialog:

```
xmlGenerator.sh -cfgFile myConfigurationFile.xml
```

The script can be integrated in an external batch process launched from the command line. The command line parameter of the script is the relative path to the exported XML settings file. The files specified with relative paths in the exported XML settings will be made absolute relative to the folder where the script is run.

The following example shows such an XML configuration file:

XML Configuration File

```
<settings>

<schemaSystemId>http://www.w3.org/2001/XMLSchema.xsd</schemaSystemId>

    <documentRoot>schema</documentRoot>
    <outputFolder>D:\projects\output</outputFolder>
    <filenamePrefix>instance</filenamePrefix>
```

```
    <filenameExtension>xml</filenameExtension>
    <noOfInstances>1</noOfInstances>
    <openFirstInstance>true</openFirstInstance>
    <defaultNamespace>&lt;NO_NAMESPACE></defaultNamespace>
    <element namespace="&lt;ANY>" name="&lt;ANY>">
       <generateOptionalElements>false</generateOptionalElements>


<generateOptionalAttributes>false</generateOptionalAttributes>
        <valuesForContentType>DEFAULT</valuesForContentType>

<preferredNumberOfRepetitions>2</preferredNumberOfRepetitions>
        <maximumRecursivityLevel>1</maximumRecursivityLevel>
        <choicesAndSubstitutions strategy="RANDOM"
                generateOthersAsComments="false"/>
        <attribute namespace="&lt;ANY>"
                name="&lt;ANY>">
            <attributeValue>attrValue1</attributeValue>
            <attributeValue>attrValue2</attributeValue>
        </attribute>
    </element>
    <element namespace="&lt;NO_NAMESPACE>"
            name="&lt;ANY>">
        <generateOptionalElements>true</generateOptionalElements>

<generateOptionalAttributes>true</generateOptionalAttributes>
        <valuesForContentType>DEFAULT</valuesForContentType>

<preferredNumberOfRepetitions>2</preferredNumberOfRepetitions>
        <maximumRecursivityLevel>1</maximumRecursivityLevel>
        <choicesAndSubstitutions strategy="RANDOM"
                generateOthersAsComments="true"/>
        <elementValue>value1</elementValue>
        <elementValue>value2</elementValue>
        <attribute namespace="&lt;ANY>"
                name="&lt;ANY>">
            <attributeValue>attrValue1</attributeValue>
            <attributeValue>attrValue2</attributeValue>
        </attribute>
    </element>
</settings>
```

XML Schema Regular Expressions Builder

The XML Schema regular expressions builder allows testing regular expressions on a fragment of text as they are applied to an XML instance document. Start the tool from menu **Tools** > **XML Schema Regular Expressions Builder** .

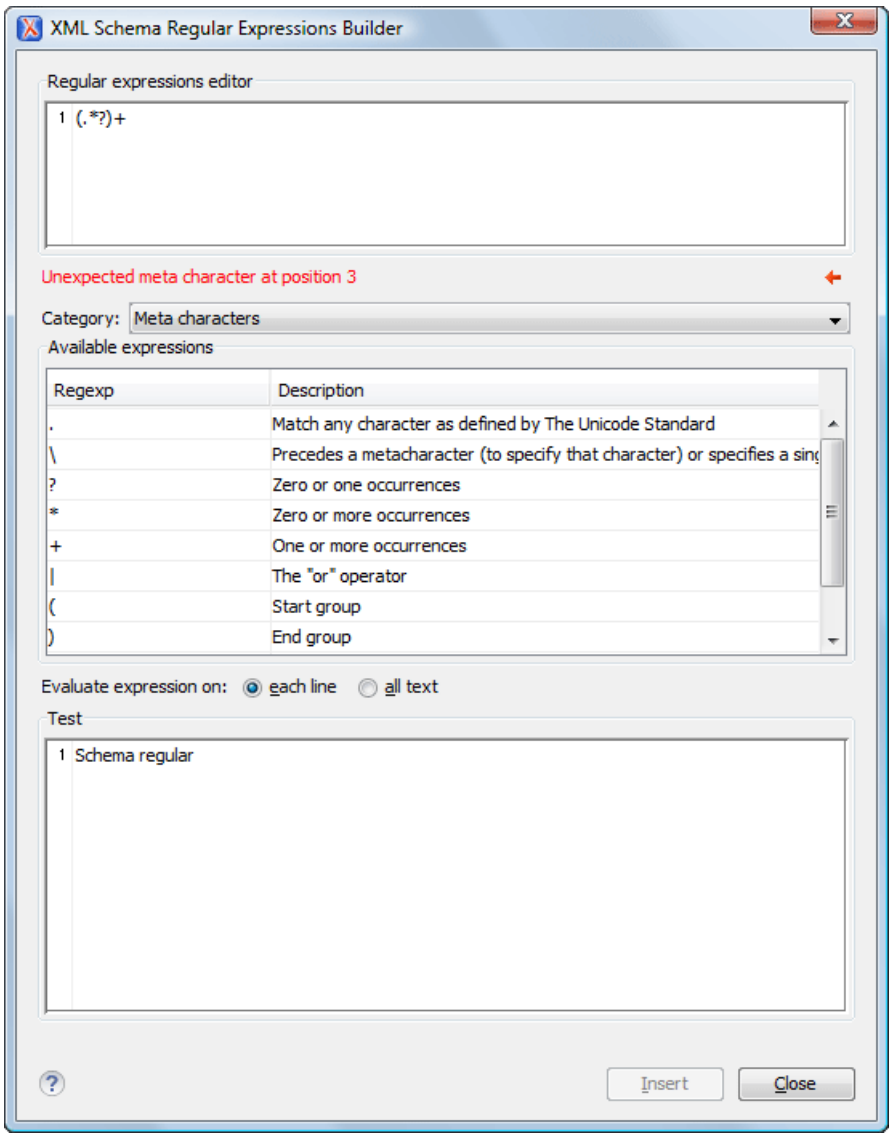


Figure 67: XML Schema Regular Expressions Builder Dialog

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 - 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 can generate detailed documentation for the components of an XML Schema in HTML, PDF and DocBook XML formats similar with the Javadoc documentation for the components of a Java class. You can select the components and the level of detail. The components are hyperlinked in both HTML and DocBook documents.

To generate documentation for an XML Schema document use the dialog **Schema Documentation**. It is opened with the action **Tools** > **Generate Documentation** > **Schema Documentation... (Ctrl+Alt+S)** . It can be also opened from the **Project** view contextual menu: **Generate Documentation** > **Schema Documentation...** The dialog enables the user to configure a large set of parameters for the process of generating the documentation.

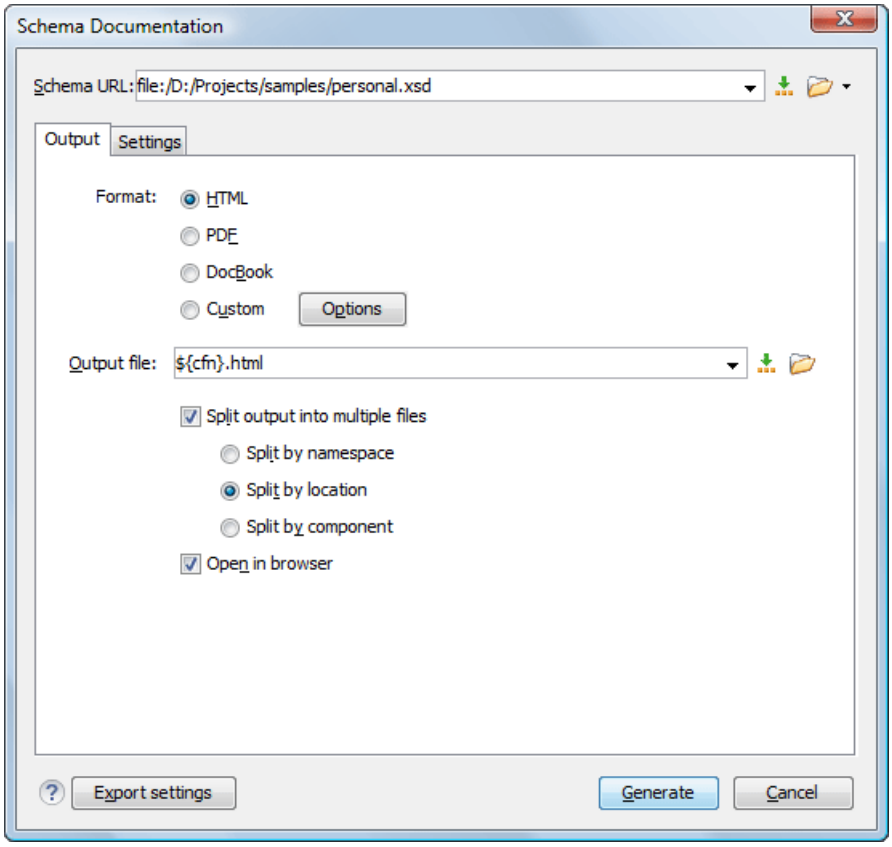


Figure 68: The Output panel of the Schema Documentation Dialog

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 proving 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** - Opens the result in the *Default Internet browser*.
- **Keep only the annotations with xml:lang set to** - The generated output will contain only the annotations with the `xml:lang` attribute set to the selected language. If you choose a primary language code (like **en**, for example), this includes all its possible variations (for example **en-us** and **en-uk** just to name a few).

You can choose to split the output into multiple files by namespace, location or component.

You can export the settings of the 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*.

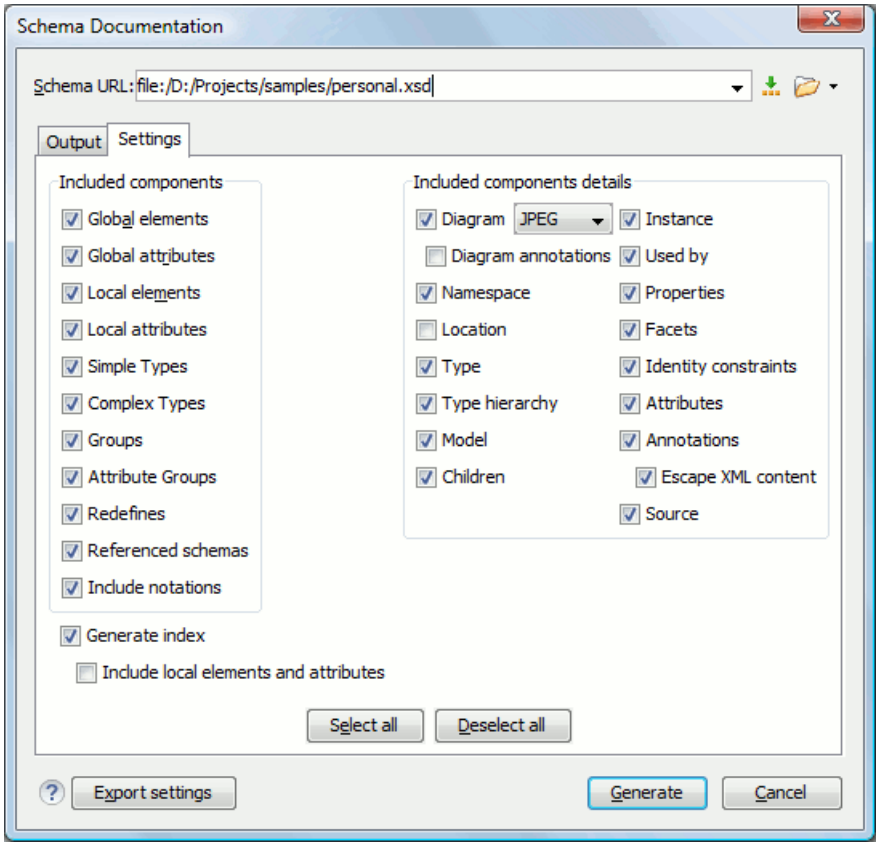


Figure 69: The Settings Panel of the Schema Documentation Dialog

When you generate documentation for a schema you can choose what components to include in the output (global elements, global attributes, local elements, local attributes, simple types, complex types, group, attribute groups, referenced schemas, redefines) and the details to be included in the documentation:

- **Diagram** - Displays the diagram for each component. You can choose the image format (JPEG, PNG, GIF, SVG) to use for the diagram section.
- **Diagram annotations** - This option controls whether or not the annotations of the components presented in the diagram sections should be 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.
- **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.

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 `<b>`, `<i>`, `<u>`, `<ul>`, `<li>`, 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's XSD editor panel.*

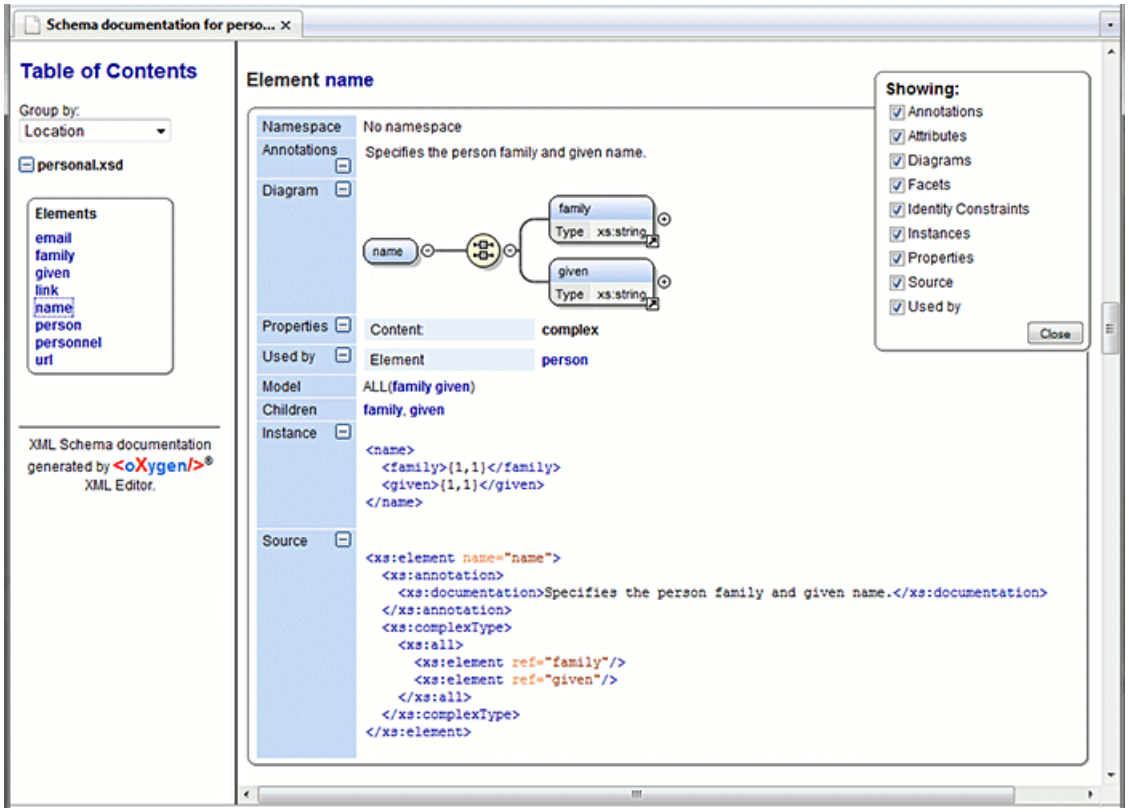


Figure 70: Schema Documentation Example

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.

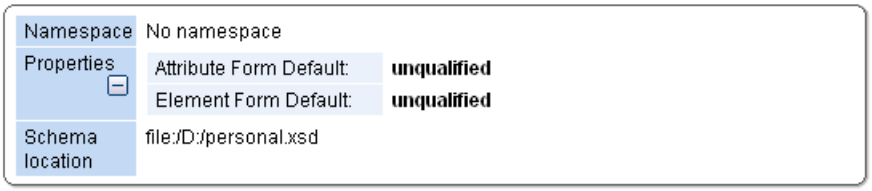


Figure 71: Information About a Schema

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:

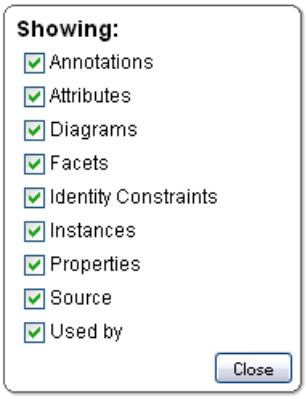


Figure 72: The Showing View

For each component included in the documentation, the section presents the component type follow by the component name. For local elements and attributes, the name of the component is specified as *parent name/component name*. You can easily go to the parent documentation by clicking the parent name.

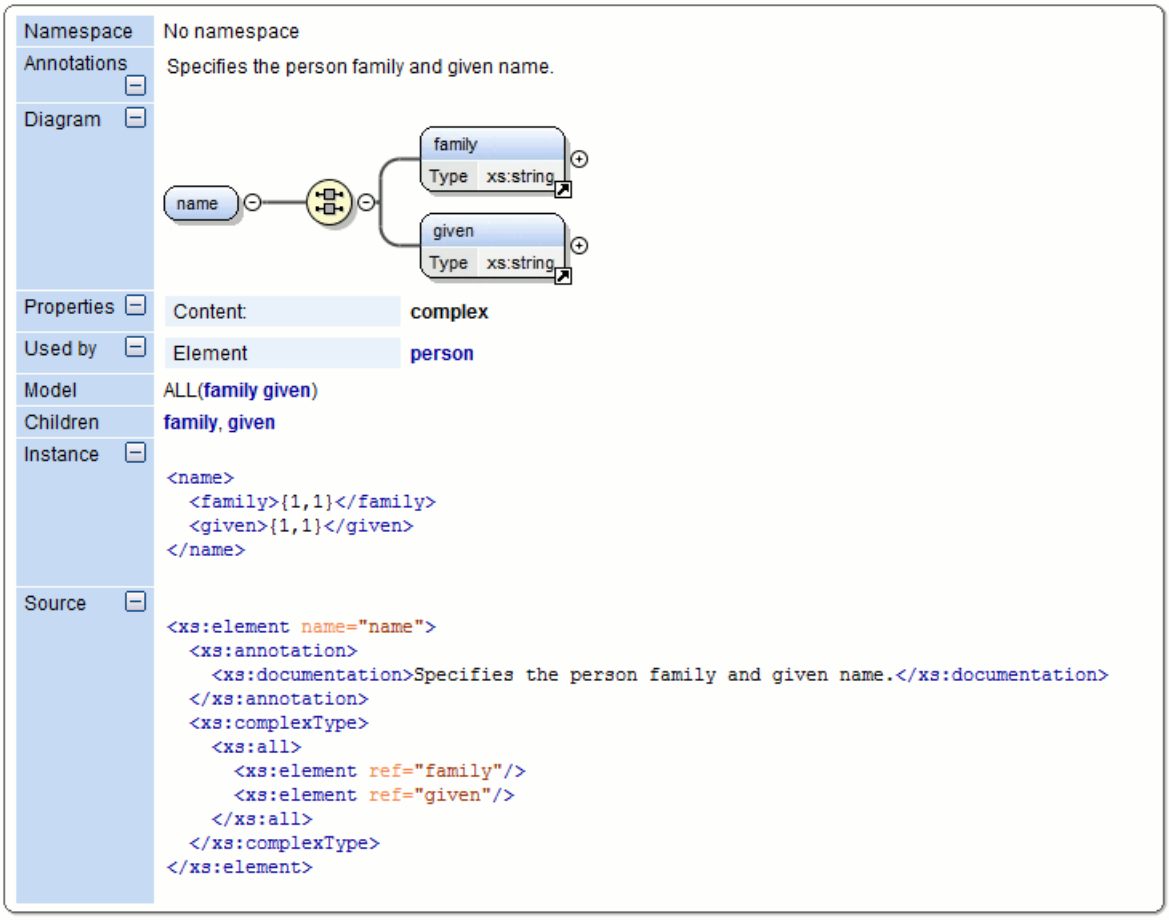
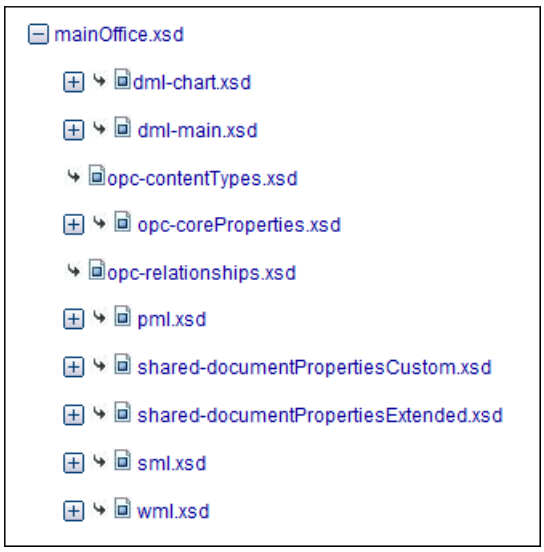


Figure 73: Documentation for a Schema Component

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

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 (*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.

Generating Documentation From the Command-Line Interface

You can export the settings of the **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 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>
                <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:

- **Document** > **References** >  > **References (Ctrl+Shift+R)** - 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.
- **Document** > **References** > **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.
- **Document** > **References** >  > **Declarations (Ctrl+Shift+D)** - 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** page.

- **Document** > **References** > **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** page.
- **Document** > **References** > **Occurrences in File (Ctrl+Shift+U)** - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor** > **Rename Component...** - Renames the selected component. Specify the new component name and the file or files affected by the modification in the following dialog:

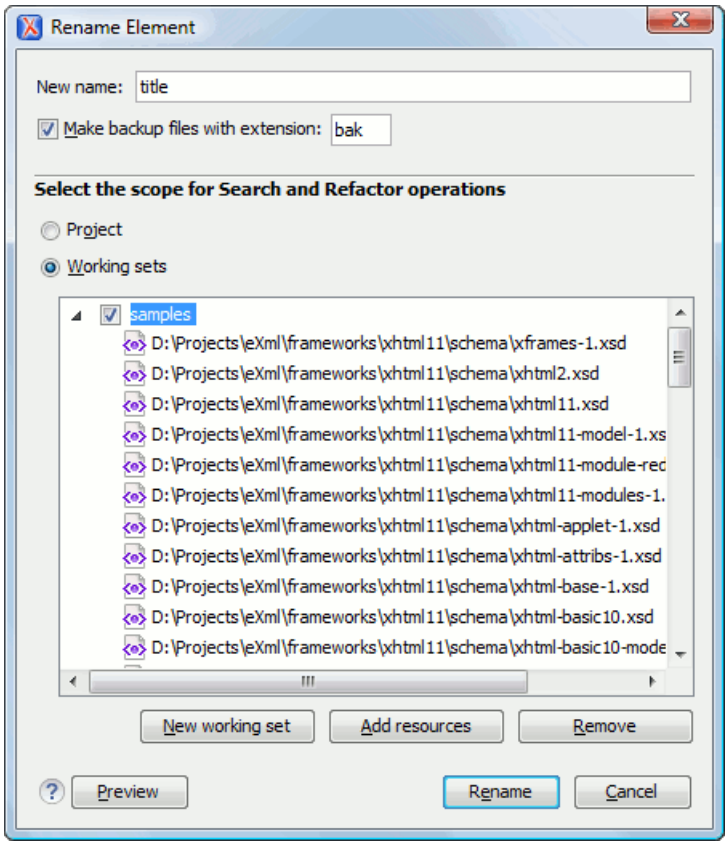


Figure 74: Rename Component Dialog

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:

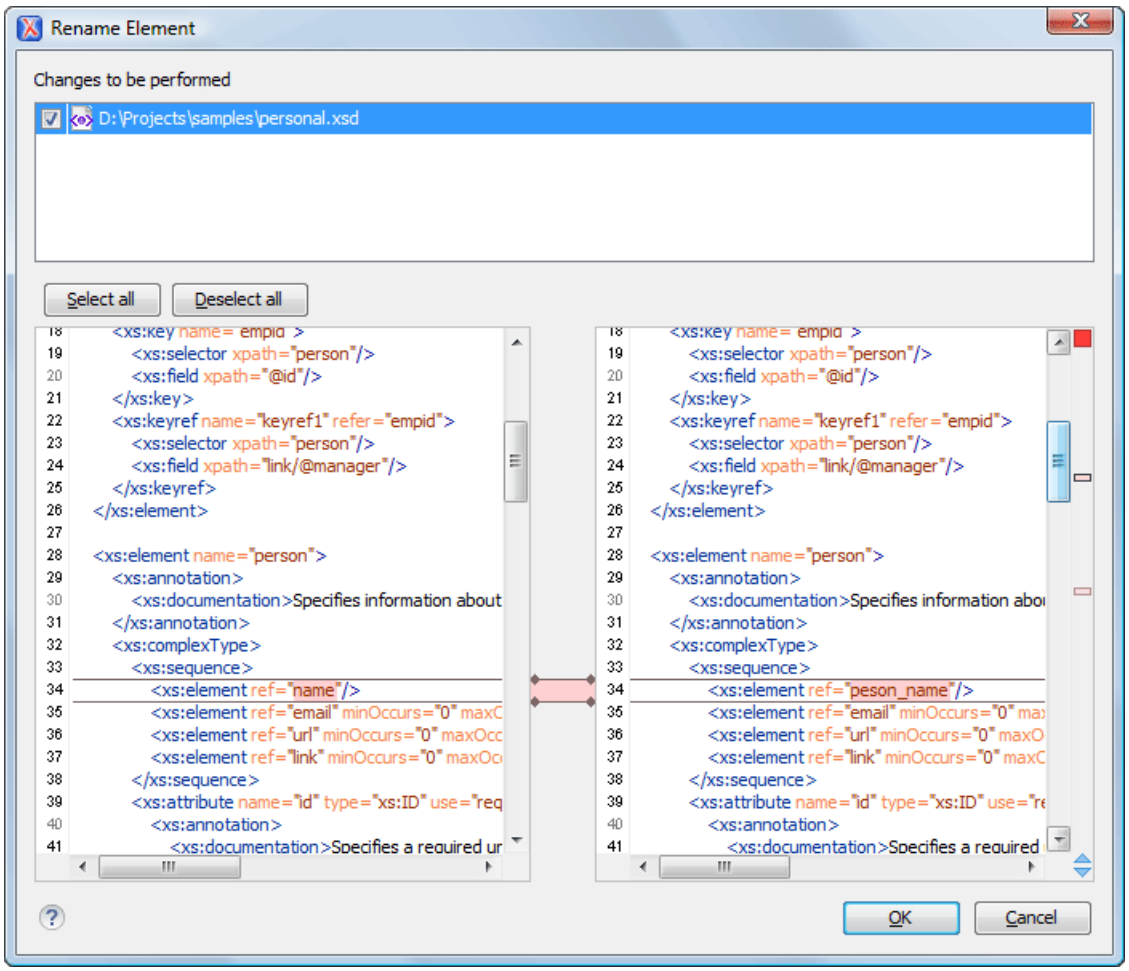


Figure 75: Preview Dialog

Search and Refactor Operations Scope

By *scope* you should understand the collection of documents that define the context of the search and refactor operations. To control it you can use the **Change scope** operation, available in the Quick Assist action set or on the **Resource Hierarchy/Dependency View** toolbar. Here you can restrict the scope to the current project or to one or multiple working sets.

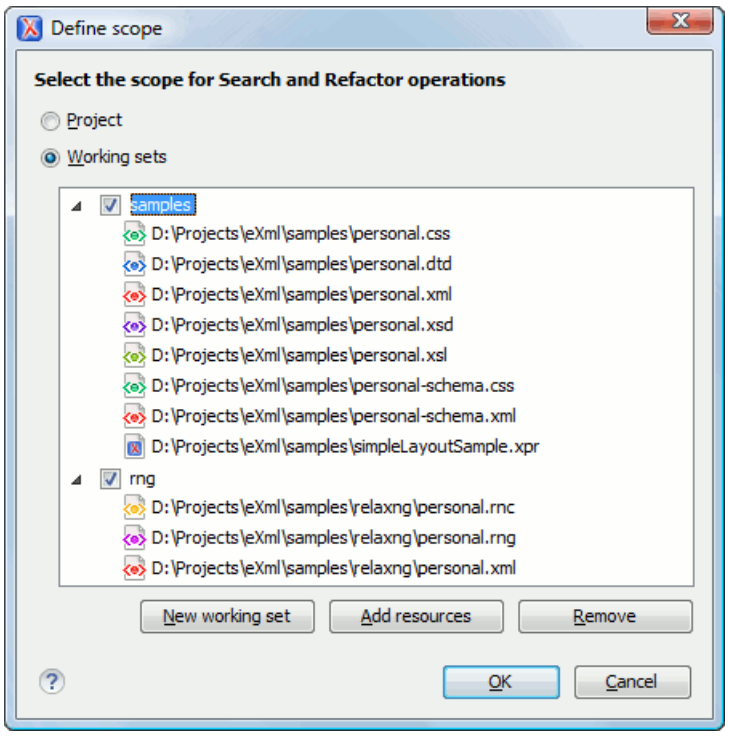


Figure 76: Change Scope Dialog

The defined scope is applied to all future search and refactor operations until you alter it again. Contextual menu actions allow you to add or delete files, folders and other resources to the working set structure.

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*. You can open the view from **Window** > **Show View** > **Resource Hierarchy / Dependencies** .

This view is useful for example 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. Also the same view is able to build the inversed-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 Oxygen 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.

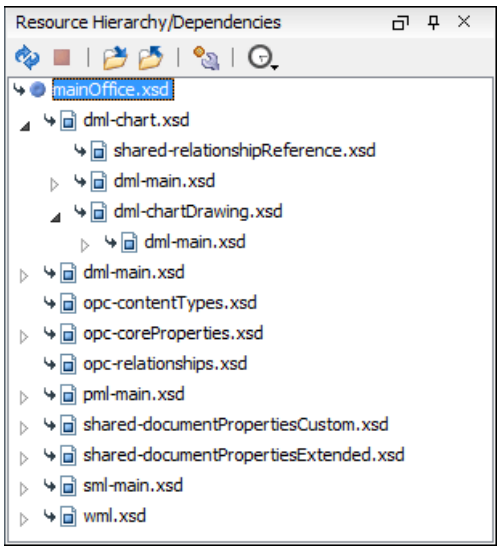


Figure 77: Resource Hierarchy/Dependencies View - Hierarchy for `mainOffice.xsd`

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

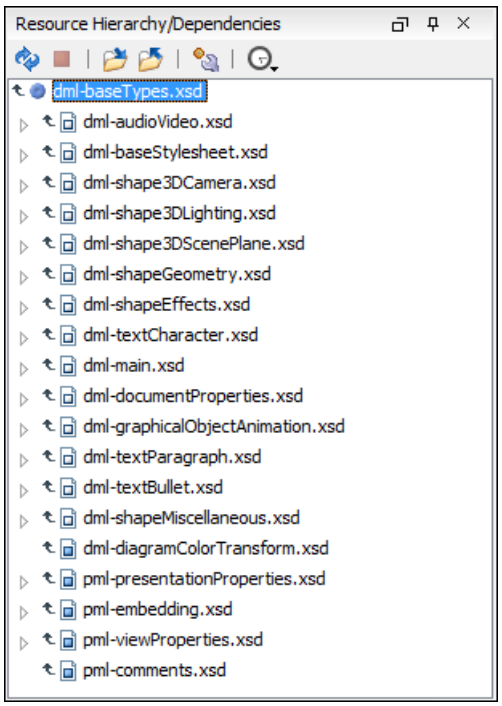


Figure 78: Resource Hierarchy/Dependencies View - Dependencies for `dml-baseTypes.xsd`

In the **Resource Hierarchy/Dependencies** view you have several actions in the toolbar:

- - Refreshes the Hierarchy/Dependencies structure.
- - Stop the hierarchy/dependencies computing.
- - Allows you to choose a schema to compute the hierarchy structure.
- - Allows you to choose a schema 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 schema. Also you can open the schema by a double-click on the Hierarchy/Dependencies structure.
- **Copy location** - Copies the location of the schema.
- **Show Resource Hierarchy** - Shows the hierarchy for the selected schema.
- **Show Resource Dependencies** - Shows the dependencies for the selected schema.
- **Expand All** - Expands all the children of the selected schema from the Hierarchy/Dependencies structure.
- **Collapse All** - Collapses all children of the selected schema from the Hierarchy/Dependencies structure.

Tip: When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .

Component Dependencies View

The **Component Dependencies** view allows you to spot the dependencies for the selected component of an XML Schema, a *Relax NG schema*, a *NVDL schema* or an *XSLT stylesheet*. You can open the view from **Window** > **Show View** > **Component Dependencies** .

If you want to see the dependencies of a schema component:

- select the desired schema component in the editor;
- choose the **Component Dependencies** action from the contextual menu.

The action is available for all named components (for example elements or attributes).

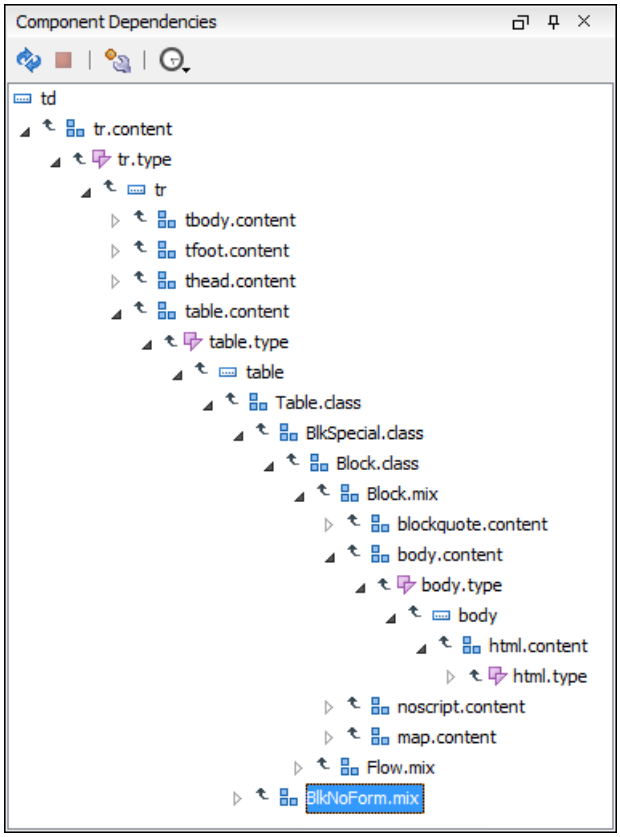


Figure 79: Component Dependencies View - Hierarchy for `xhtml11.xsd`

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 .

Highlight Component Occurrences

When a component (for example types, elements, attributes) is found at current cursor position, Oxygen 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+Shift+U)** contextual menu action. All matches are displayed in a separate tab of the **Results** view.

XML Schema Quick Assist Support

The Quick Assist action set was designed to help you improve the development work flow by offering quicker access to the most commonly used actions.

It is activated automatically when the cursor is positioned inside a component name. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X).

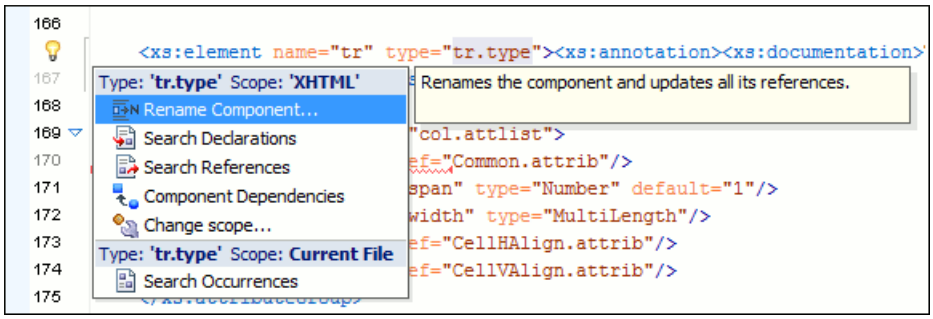


Figure 80: Quick Assist Support

The quick assist support offers direct access to the following actions:

- **Rename Component** - Renames the component and all its dependencies;
- **Search Declaration** - 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;
- **Search Occurrences** - Searches all occurrences of the file within the current scope.

Linking Between Development and Authoring

The Author page is available on the XML Schema editor allowing you to edit visually the schema annotations. It presents a polished and compact view of the XML Schema, with support for links on imported/included schemas. Embedded Schematron is supported only in Relax NG schemas with XML syntax.

Editing Relax NG Schemas

Oxygen 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.

Relax NG Schema Diagram

This section explains how to use the graphical diagram of a Relax NG schema.

Introduction

Oxygen 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 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 underlaying 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**.

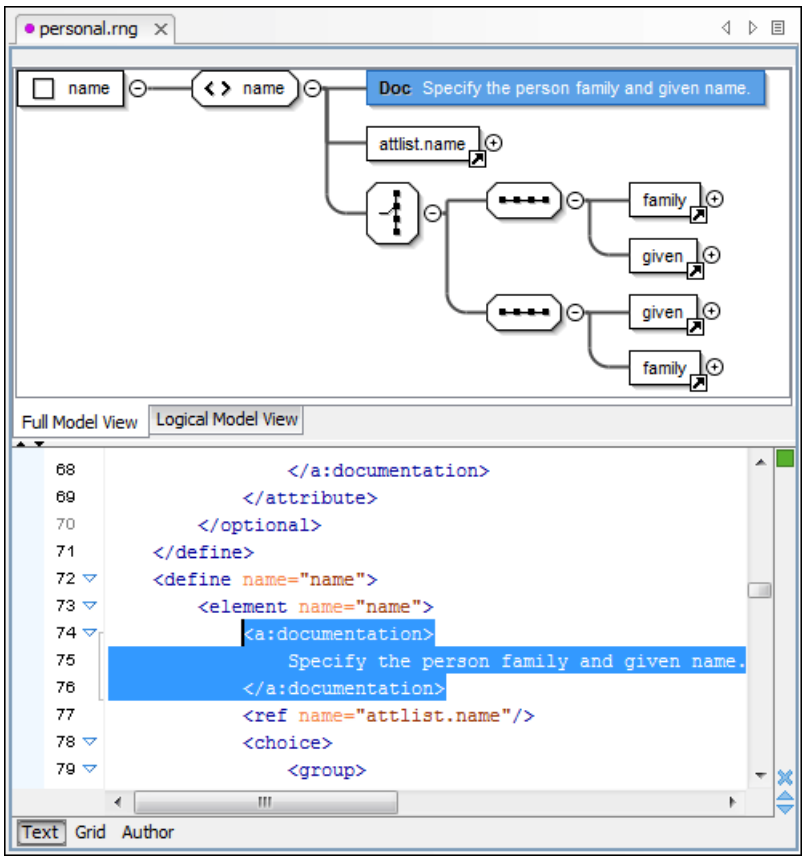


Figure 81: Relax NG Schema Editor - Full Model View

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.

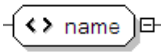
Symbols Used in the Schema Diagram

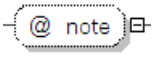
The **Full Model View** renders all the Relax NG Schema patterns with intuitive symbols:

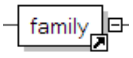
- 

- `define` pattern with the `name` attribute set to the value shown inside the rectangle (in this example `name`).
- 

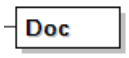
- `define` pattern with the `combine` attribute set to `interleave` and the `name` attribute set to the value shown inside the rectangle (in this example `attlist.person`).
- 

- `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 de 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.

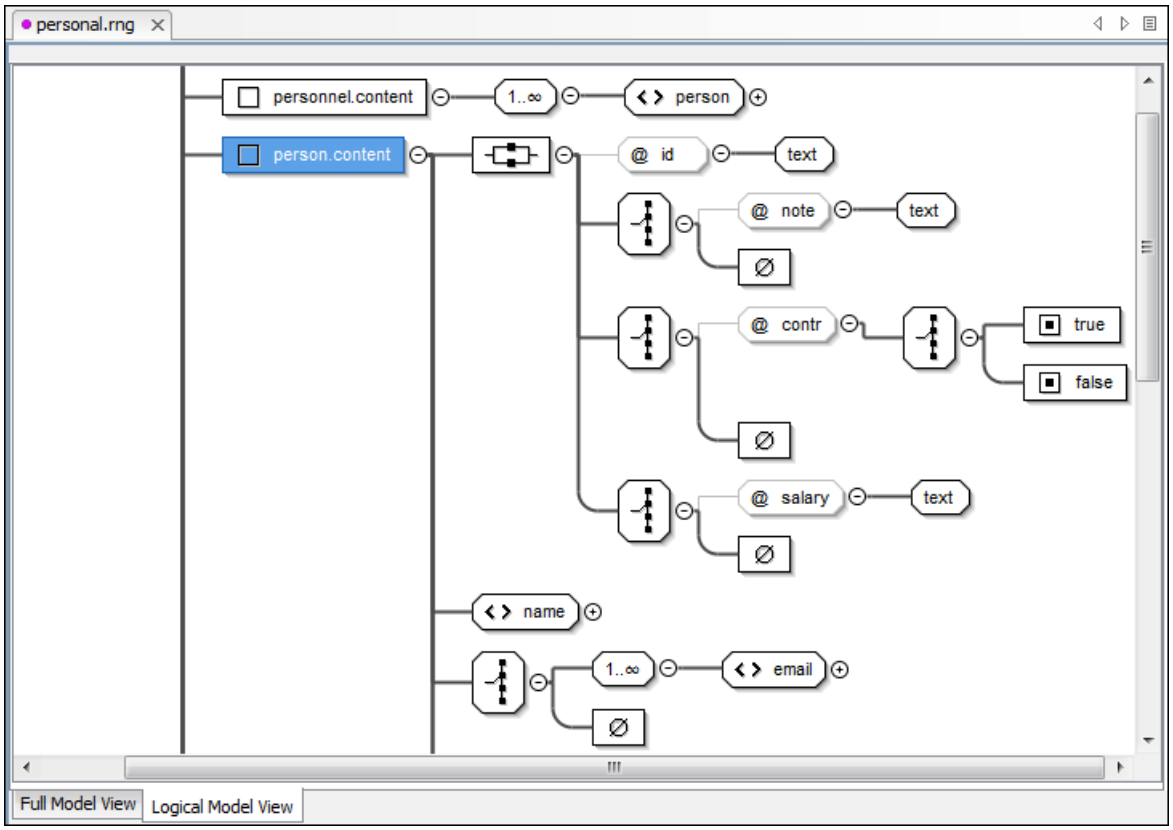


Figure 82: Logical Model View for a Relax NG Schema

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

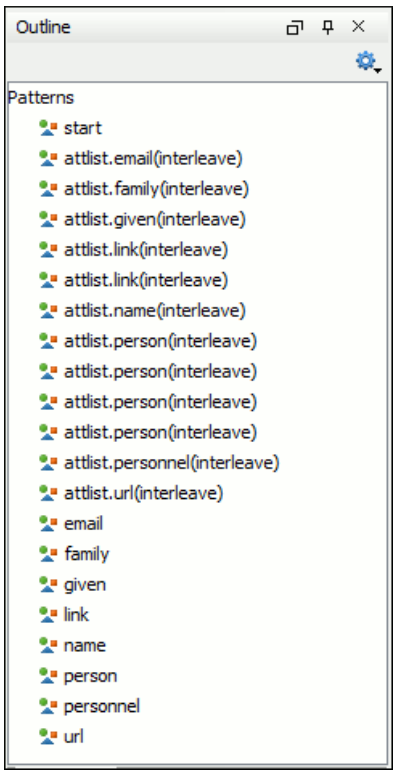


Figure 83: Relax NG Outline View

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 **Settings** menu on the Outline view's toolbar:

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

When the XML elements are displayed, the following actions are available in the **Settings** menu on the Outline view's toolbar:

- **Selection update on caret move** - Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram will be also selected in the **Outline** view.
- **Show components** - Shows the define patterns collected from the current document.
- **Flat presentation mode of the filtered results** - when active, the application flattens the filtered result elements to a single level.
- **Show comments and processing instructions** - show/hide comments and processing instructions in the **Outline** view.
- **Show 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:

- **Document** > **Schema** > **Show Definition** (also available on the contextual menu of the editor panel) - Moves the cursor to the definition of the current element in this Relax NG (full syntax) schema.

Searching and Refactoring Actions

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

- **Document** > **References** >  **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.

   **Note:** This action and the following ones can also be accessed from RNG editor's **contextual menu** > **Search** .

  You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.
- **Document** > **References** > **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.

- **Document** > **References** >  **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.

   **Note:** This action and the following ones can also be accessed from RNG editor's **contextual menu** > **Search** .

- **Document** > **References** > **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the files specified in the search scope.
- **Document** > **References** > **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.

Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a schema. You can open the view from **Window** > **Show View** > **Resource Hierarchy/Dependencies** .

If you want to see the hierarchy of a schema, select the desired schema in the project view and choose **Resource Hierarchy** from the contextual menu.

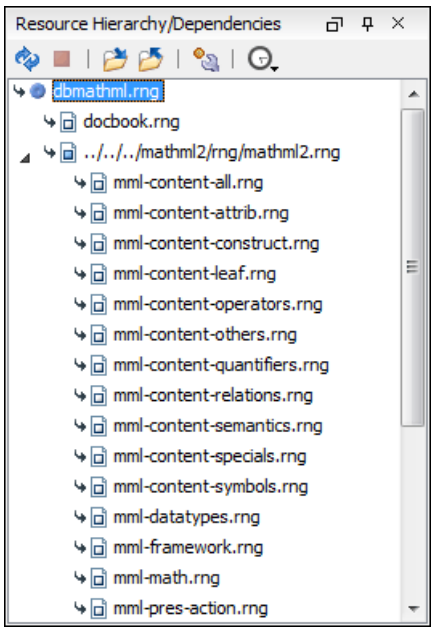


Figure 84: Resource Hierarchy/Dependencies View - hierarchy for `dbmathml.rng`

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.

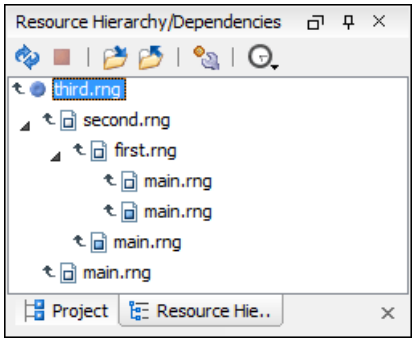


Figure 85: Resource Hierarchy/Dependencies View - Dependencies for `third.rng`

In the **Resource Hierarchy/Dependencies** view you have several actions in the toolbar:

- - Refreshes the Hierarchy/Dependencies structure.
- - Stops the hierarchy/dependencies computing.
- - Allows you to choose a schema to compute the hierarchy structure.
- - Allows you to choose a schema to compute the dependencies structure.
- - Allows you to configure a scope to compute the dependencies structure.
- - Repeats a previous dependencies computation.

The following actions are available in the contextual menu:

- **Open** - Opens the schema. Also you can open the schema by a double-click on the Hierarchy/Dependencies structure.
- **Copy location** - Copies the location of the schema.
- **Resource Hierarchy** - Shows the hierarchy for the selected schema.
- **Resource Dependencies** - Shows the dependencies for the selected schema.
- **Expand All** - Expands all the children of the selected schema from the hierarchy/dependencies structure.
- **Collapse All** - Collapses all the children of the selected schema from the hierarchy/dependencies structure.


Tip: When a recursive reference is encountered in the **Hierarchy** view, the reference is marked with a special icon  .

Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected Relax NG component. You can open the view from **Window** > **Show View** > **Component Dependencies** .

If you want to see the dependencies of a RelaxNG component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named defines.

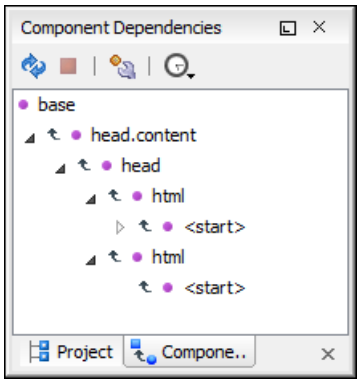


Figure 86: Component Dependencies View - Hierarchy for `base.rng`

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 Assist action set was designed to help you improve the development work flow by offering quicker access to the most commonly used actions.

It is activated automatically when the cursor is positioned inside a component name. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X).

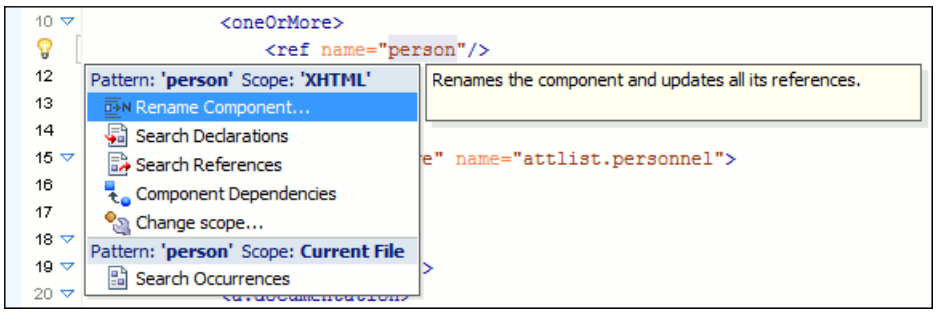


Figure 87: RNG Quick Assist Support

The quick assist support offers direct access to the following actions:

- **Rename Component** - Renames the component and all its dependencies;
- **Search Declaration** - 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;
- **Search Occurrences** - Searches all occurrences of the file within the current scope.

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.*

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-install-folder]/lib` .

To load the custom library, restart Oxygen.

Linking Between Development and Authoring

The Author page 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 which 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 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 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 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 underlaying 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**.

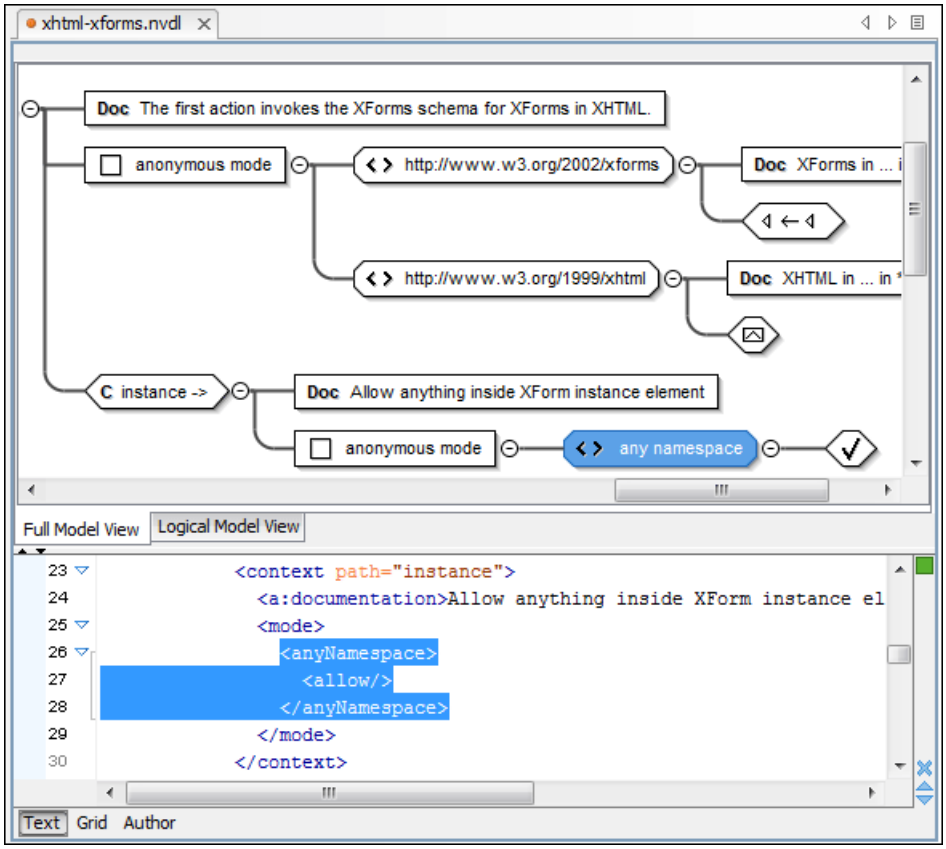


Figure 88: NVDL Schema Editor - Full Model View

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** > **Outline** menu.

NVDL Editor Specific Actions

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

- **Document** > **Schema** > **Show Definition** (also available on the contextual menu of the editor panel) - Moves the cursor to its definition in the schema used by NVDL in order to validate it.

Searching and Refactoring Actions

All the following actions can be applied on mode `name`, `useMode`, and `startMode` attributes only.

- **Document** > **References** > **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.

  Note: This action and the following ones can also be accessed from NVDL editor's **contextual menu** > **Search** .

  You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.
- **Document** > **References** > **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.

- **Document** > **References** > **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.

  Note: This action and the following ones can also be accessed from NVDL editor's **contextual menu** > **Search** menu.

- **Document** > **References** > **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files specified in the search scope.
- **Document** > **References** > **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** > **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.

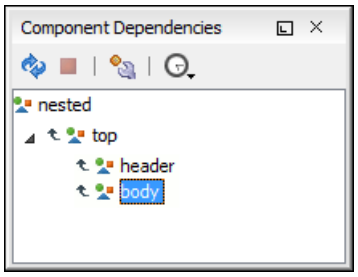


Figure 89: Component Dependencies View - Hierarchy for `test.nvdl`

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 page is available on the NVDL scripts editor presenting them in a compact and easy to understand representation.

Editing XSLT Stylesheets

This section explains the features of the XSLT editor.

Validating XSLT Stylesheets

Validation of XSLT stylesheets documents is performed with the help of an XSLT processor *configurable from user preferences* according to the XSLT version: 1.0 or 2.0. For XSLT 1.0, the options are: Xalan, Saxon 6.5.5, Saxon 9.3.0.5, MSXML 4.0, MSXML.NET, *a JAXP transformer specified by the main Java class.* For XSLT 2.0, the options are: Saxon 9.3.0.5, *a JAXP transformer specified by the main Java class.*

The **Validate** toolbar provides a button **Validation options** for quick access to the *XSLT options* page in the Oxygen user preferences.

Custom Validation of XSLT Stylesheets

If you must validate an XSLT stylesheet with other validation engine than the Oxygen'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. The document is validated against the schema declared in the document.

There are two validators configured by default:

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

Associate a Validation Scenario

Validation of XSLT stylesheets documents can be also performed through a validation scenario. To define a validation scenario, open the **Configure Validation Scenario** dialog. You do this with the **Configure Validation Scenario** action available on the menu **Document** > **Validate** and on the **Validate** toolbar. .

You can validate an XSLT document using the engine from 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 **Options** > **Preferences** > **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 see *Validation Scenario*.

Contextual Editing

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 provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included files in the context of the larger stylesheet structure.

To set a main files, you need to define a validation scenario and add validation units that point to the main modules. Oxygen 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.

Content Completion in XSLT Stylesheets

Inside XSLT templates of an XSLT stylesheet, the content completion assistant presents all the elements allowed in any context by the schema associated to the result of applying the stylesheet. That schema is *defined by the user in the Content Completion / XSL preferences* page and can be of type: XML Schema, DTD, RELAX NG schema, or NVDL

schema. There are presented all the elements because in a template there is no context defined for the result document. The user is allowed to insert any element defined by the schema of the result document.

The content completion window lists 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

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 or *http://icl.com/saxon* for XSLT version 1.0) is mapped to a prefix 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 9.3.0.5 PE or Saxon 9.3.0.5 EE (for XSLT version 2.0);
- the edited file has a validation scenario that uses as validation engine Saxon 6.5.5 (for version 1.0), Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE (for version 2.0);
- the validation engine specified in *Options* page is Saxon 6.5.5 (for version 1.0), Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE (for version 2.0).

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.

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), Oxygen provides an easy mode to map them by proposing a prefix for these namespaces.

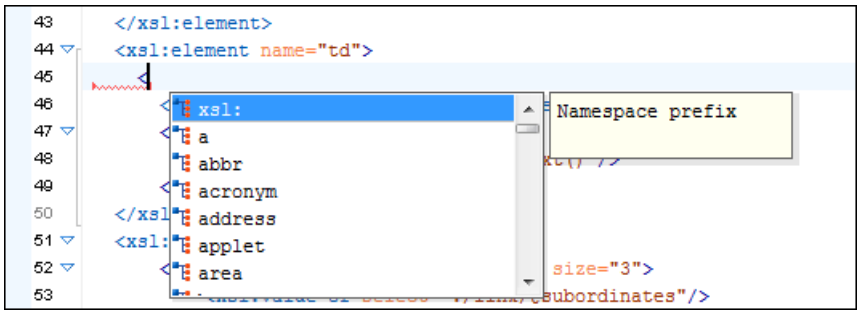


Figure 90: Namespace Prefixes in the Content Completion Window

Content Completion in XPath Expressions

In XSLT stylesheets, the content completion assistant provides *all the features available in the XML editor* and also adds some enhancements. In XPath expressions used in attributes of XSLT stylesheets elements like `match`, `select` and `test`, the content completion assistant offers the names of XPath and XSLT functions, the XSLT axes, and user-defined functions (the name of the function and its parameters). If a transformation scenario was defined and associated to the edited stylesheet, the content completion assistant computes and presents elements and attributes based on:

- the input XML document selected in the scenario;
- the current context in the stylesheet.

The associated document is displayed in *the **XSLT/XQuery Input** view*.

Content completion for XPath expressions is started:

- on XPath operators detected in one of the `match`, `select` and `test` attributes of XSLT elements: ", ', /, //, (, [, |, :, ::, $

- for attribute value templates of non-XSLT elements, that is the `{` character when detected as the first character of the attribute value
- on request, if the combination CTRL + Space 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 or 2.0).

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 or 2.0.

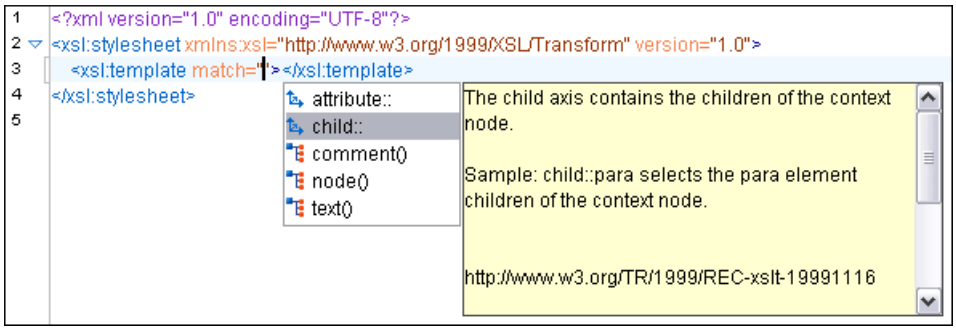


Figure 91: Content Completion in the `match` Attribute

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:

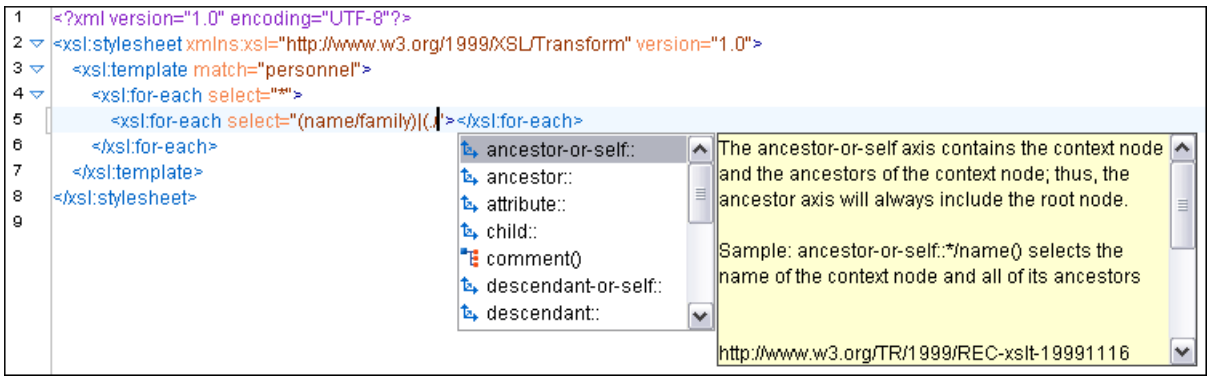


Figure 92: Content Completion in the `select` Attribute

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

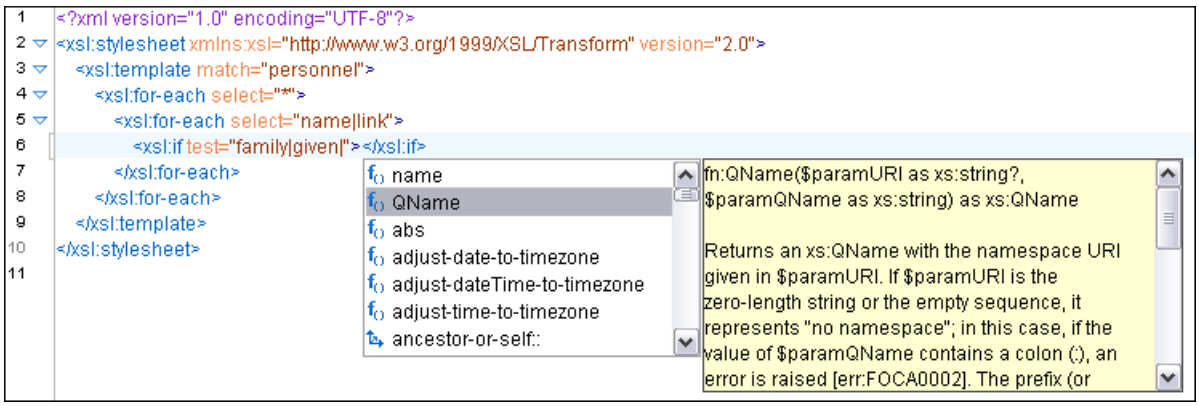


Figure 93: Content Completion in the `test` Attribute

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.

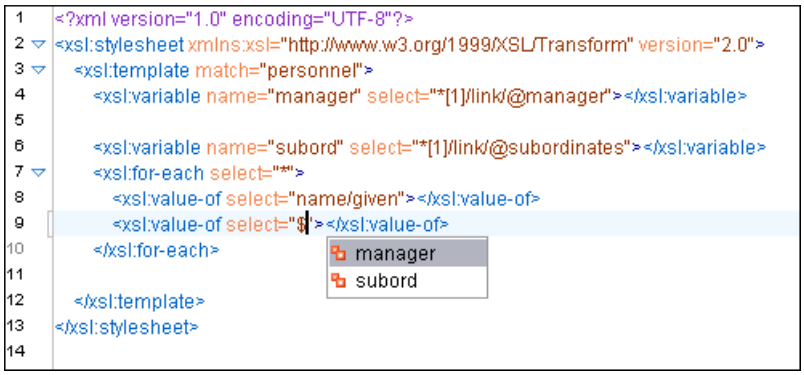


Figure 94: Content Completion in the `test` Attribute

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.

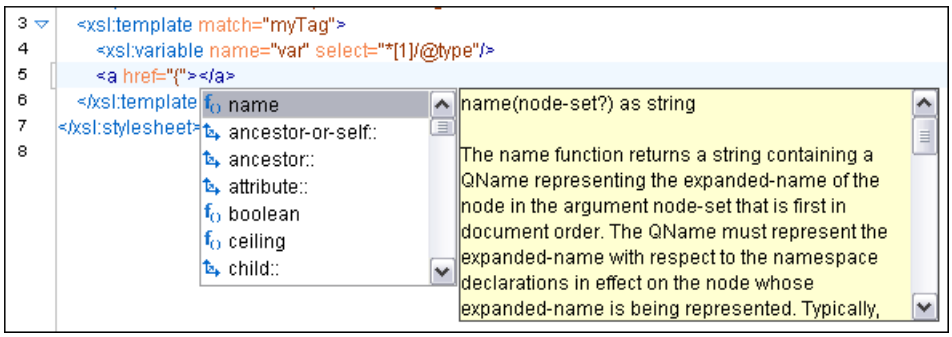


Figure 95: Content Completion in Attribute Value Templates

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 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 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`.

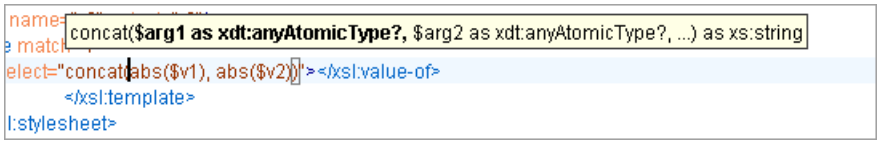


Figure 96: XPath Tooltip Helper - Identify the `concat` Function's First Argument

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

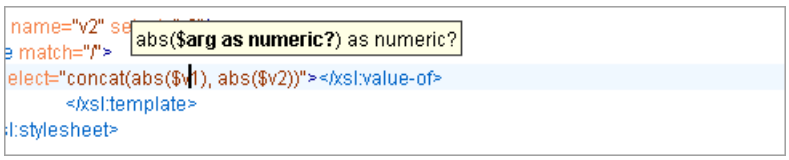


Figure 97: XPath Tooltip Helper - Identify the `abs` Function's Argument

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.

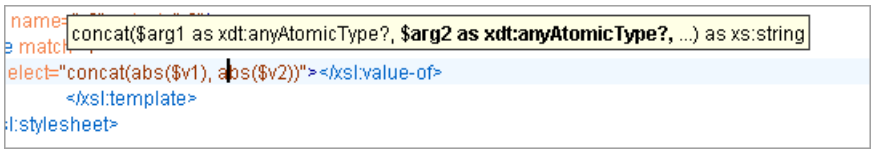


Figure 98: XPath Tooltip Helper - Identify the `concat` Function's Second Argument

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+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 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.

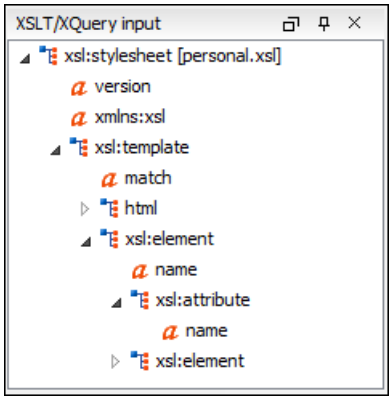


Figure 99: XSLT Input View

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:

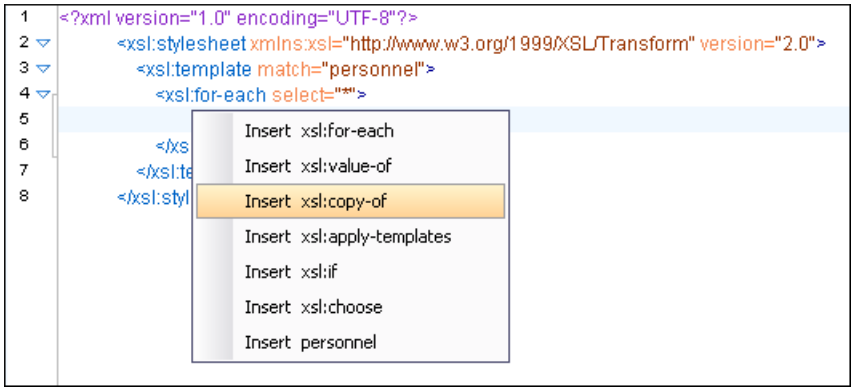


Figure 100: XSLT Input Drag and Drop Popup Menu

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

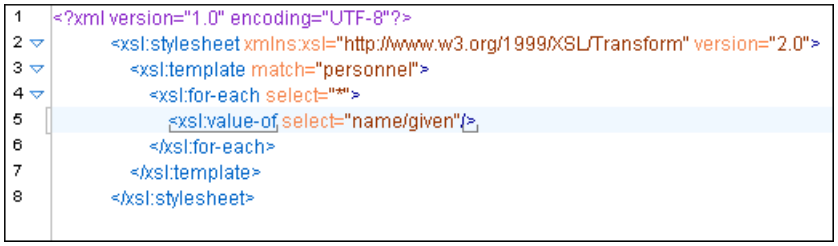


Figure 101: XSLT Input Drag and Drop Result

The XSLT Outline View

The XSLT Outline View presents the list of all the components (templates, attribute-sets, character-maps, variables, functions) from both the edited stylesheet and its imports or includes. It can be opened from menu **Window** > **Show View** > **Outline** .

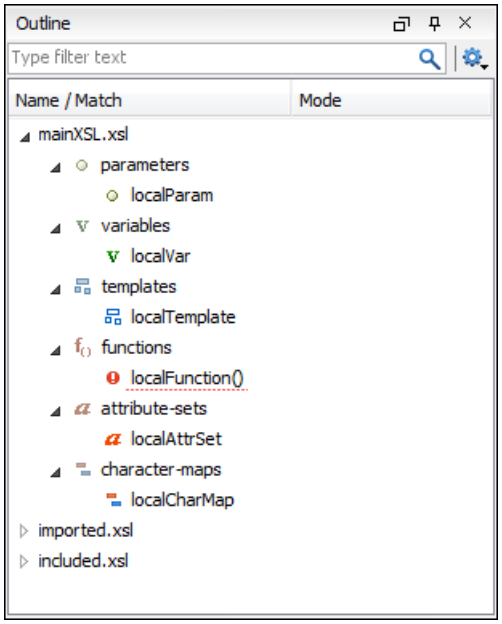


Figure 102: The XSLT Outline View

The following actions are available in the **Settings** menu on the Outline view's toolbar:

- **Selection update on caret move** - Allows a synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret's 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 document's XML structure in a tree-like structure.
- **Sort** - Alphabetically sorts the stylesheet components.
- **Show all components** - Displays all components that were collected starting from the main file. This option is set by default.
- **Show local components** - Displays components defined in the current file only.
- **Group by location/type/mode** - The stylesheet components can be grouped by location, type, and mode.
- **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 text** - show/hide additional text content for the displayed elements.
- **Show attributes** - show/hide attribute values for the displayed elements. The displayed attribute values can be changed from *the Outline preferences panel*.
- **Configure displayed attributes** - displays the *XML Structured Outline preferences page*.

The following contextual menu actions are available:

- **Append Child** - Displays a list of elements that can be inserted as children of the current element.
- **Insert Before** - Displays a list of elements that can be inserted as siblings of the current element, before the current element.

- **Insert After** - Displays a list of elements that can be inserted as siblings of the current element, after the current element.
- **Toggle Comment** - Comments/uncomments the currently selected element.
- **Remove (Delete)** - Removes the selected item from the stylesheet.
- **Search References (Ctrl+Shift+R)** - 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 bottom of the view or directly on the tree structure. When you type de component name, match or mode in the text field, you can switch to the tree structure using:

- keyboard arrow keys;
- **Enter** key;
- **Tab** key;
- **Shift-Tab** key combination.

To switch from tree structure to the filter text field, you can use **Tab** and **Shift-Tab**.

Tip: The search filter is case insensitive. The following wildcards are accepted:

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

If no wildcards are specified, the string to search is used as a partial match (like ***textToFind***).

On the XSLT **Outline** view you have some contextual actions like: **Edit Attributes**, **Cut**, **Copy**, **Delete**.

The **Outline** content is synchronized with **Text** view; when you click a component in the **Outline** view, its definition is highlighted in the **Text** view.

XSLT Stylesheet Documentation Support

Oxygen 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 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+Space)**, it offers elements from the XSLT documentation schema (either the built-in one or one specified by user). A contextual action for adding documentation blocks is also available for the Text mode in the editor contextual menu **(Ctrl+Alt+D)** **(META+ALT+Comma on Mac)** > **Source** > **Add component documentation** or for the Author contextual menu **(Ctrl+Alt+D)** **(META+ALT+Comma on Mac)** > **Component documentation** > **Add component documentation** . Other documentation actions available in the Author page from the Component Documentation contextual sub menu are:

- **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 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

Oxygen can generate detailed documentation for the elements (top-level elements whose names are in the XSLT namespace) of an XSLT stylesheet in HTML format. You can select the XSLT elements to include and the level of detail to present for each of them. Also the elements are hyperlinked. The user can also use custom stylesheets to obtain a custom format.

To generate documentation for an XSLT stylesheet document, use the **XSLT Stylesheet Documentation** dialog. It is opened with the **Tools** > **Generate Documentation** > **XSLT Stylesheet Documentation... (Ctrl+Alt+X)** action. It can be also opened from the **Project** view contextual menu: **Generate Documentation** > **XSLT Stylesheet Documentation...** This dialog enables the user to configure a large set of parameters used by the application to generate the documentation.

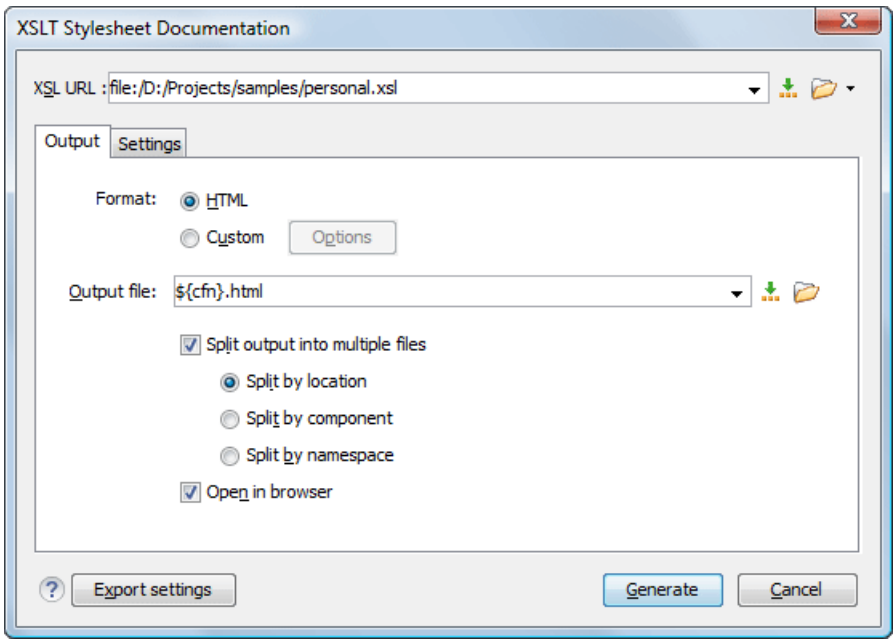


Figure 103: The Output Panel of the XSLT Stylesheet Documentation Dialog

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.

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 interface*.

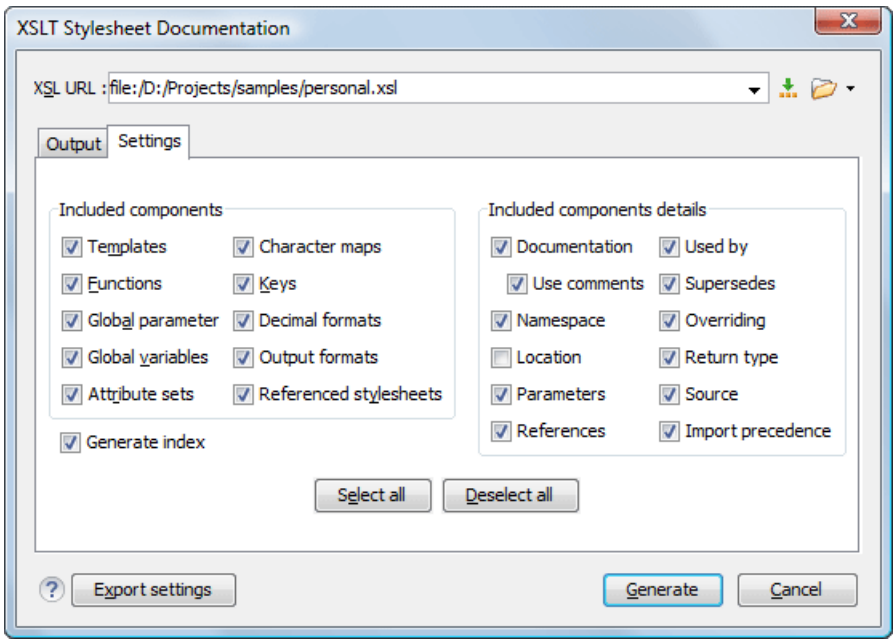


Figure 104: The Settings Panel of the XSLT Stylesheet Documentation Dialog

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 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.

Generate Documentation in HTML Format

The generated documentation looks like:

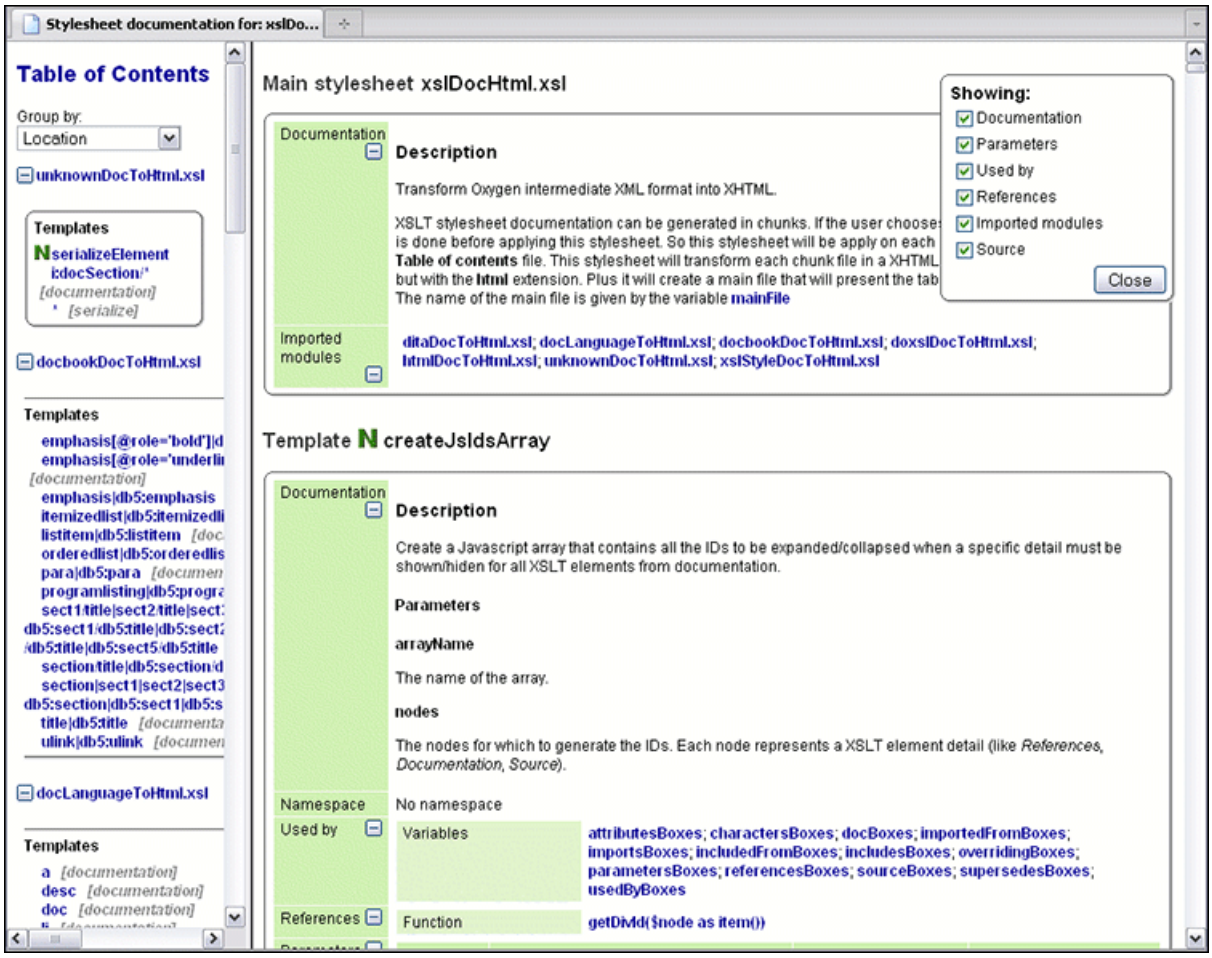


Figure 105: XSLT Stylesheet Documentation Example

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.

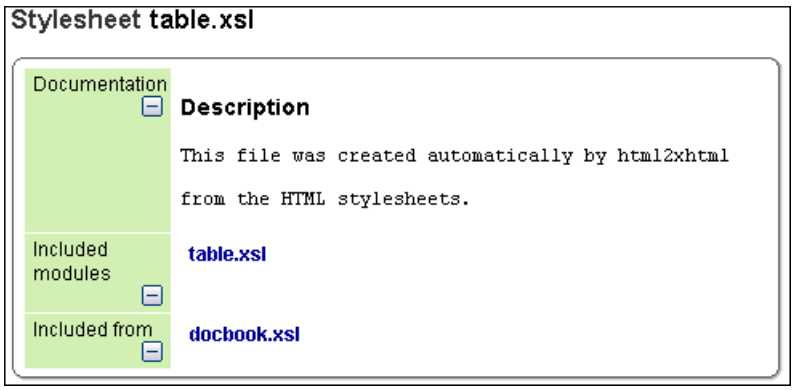


Figure 106: Information About an XSLT Stylesheet

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.

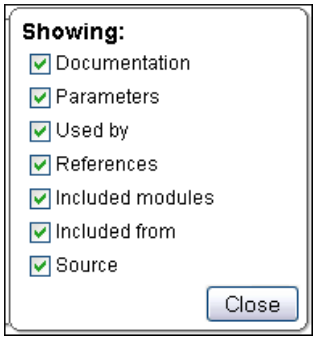


Figure 107: The Showing View

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).

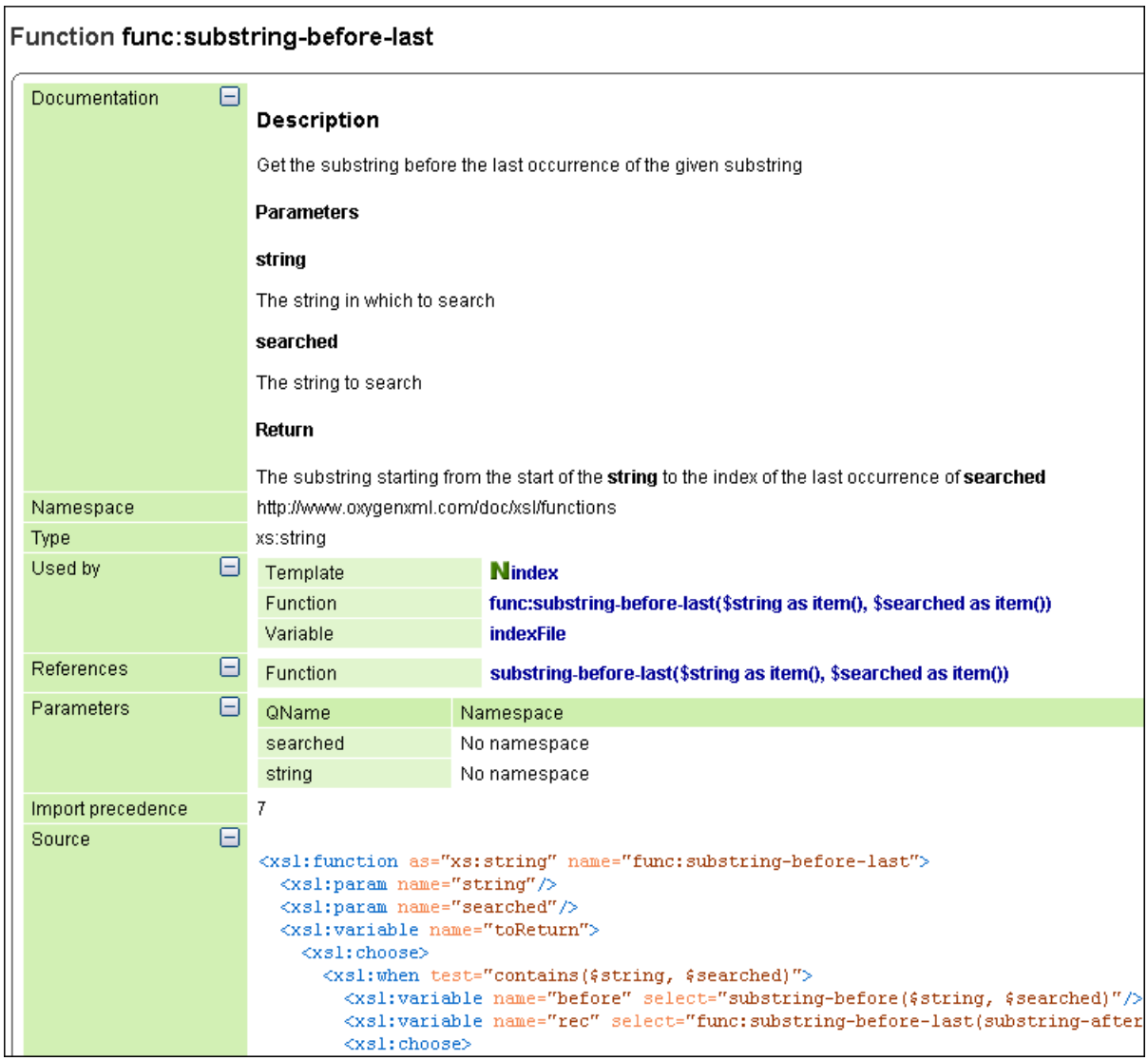


Figure 108: Documentation for an XSLT Element

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`.

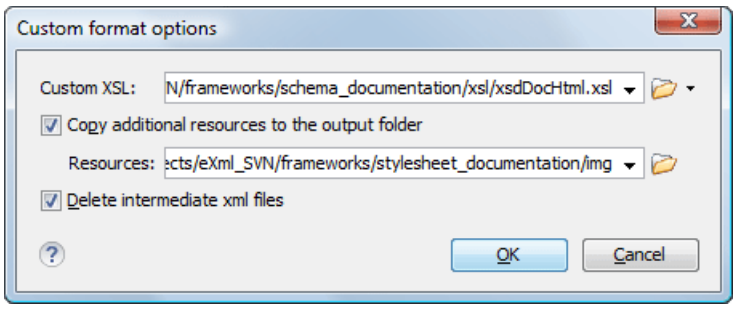


Figure 109: The Custom Format Options Dialog

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

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 installation folder. The script can be integrated in an external batch process launched from the command-line interface.

The command-line parameter of the script is the relative path to the exported XML settings file. The files which are specified with relative paths in the exported XML settings are resolved relative to the script directory.

Example of an XML Configuration File

```
<serialized>
    <map>
        <entry>
            <String
xml:space="preserve">xsd.documentation.options</String>
            <xsdDocumentationOptions>
                <field name="outputFile">
                    <String
xml:space="preserve">${cfn}.html</String>
                </field>
                <field name="splitMethod">
                    <Integer xml:space="preserve">1</Integer>
                </field>
                <field name="openOutputInBrowser">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="format">
                    <Integer xml:space="preserve">1</Integer>
                </field>
                <field name="customXSL">
                    <null/>
                </field>
                <field name="deleteXMLFiles">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeIndex">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeGlobalElements">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeGlobalAttributes">
                    <Boolean xml:space="preserve">true</Boolean>
```

```
                </field>
                <field name="includeLocalElements">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeLocalAttributes">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeSimpleTypes">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeComplexTypes">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeGroups">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeAttributesGroups">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeRedefines">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeReferencedSchemas">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsDiagram">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsNamespace">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsLocation">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsType">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsTypeHierarchy">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsModel">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsChildren">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsInstance">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsUsedby">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsProperties">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsFacets">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsAttributes">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsIdentityConstr">
```

```
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsEscapeAnn">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsSource">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="detailsAnnotations">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
            </xsdDocumentationOptions>
        </entry>
    </map>
</serialized>
```

Finding XSLT References and Declarations

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

- **Document** > **References** >  > **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.
- **Document** > **References** > **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.
- **Document** > **References** >  **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.
- **Document** > **References** > **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when define a new scope.
- **Document** > **References** > **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **Document** > **Schema** > **Show Definition** - Moves the cursor to the location of the definition of the current item.

Highlight Component Occurrences

When a component (for example variable or named template) is found at current cursor position, Oxygen 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+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:

- **Document** > **Refactoring** >  > **Create Template from Selection...** - Opens a dialog that allows the user 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.

- **Document** > **Refactoring** > > **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.

- **Document** > **Refactoring** > **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*.

Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a stylesheet. You can open the view from **Window** > **Show View** > **Resource Hierarchy/Dependencies** .

If you want to see the hierarchy of a stylesheet, select the desired stylesheet in the project view and choose **Resource Hierarchy** from the contextual menu.

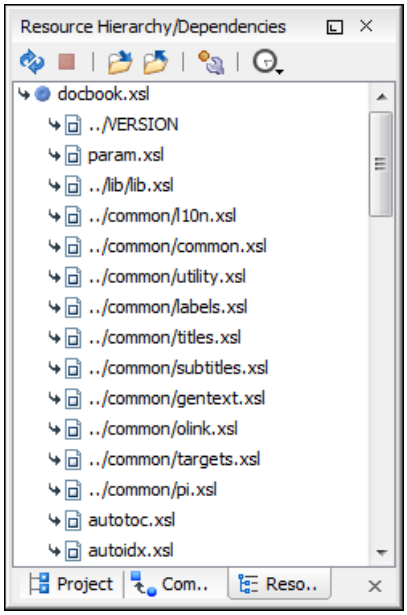


Figure 110: Resource Hierarchy/Dependencies View - Hierarchy for `docbook.xsl`

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.

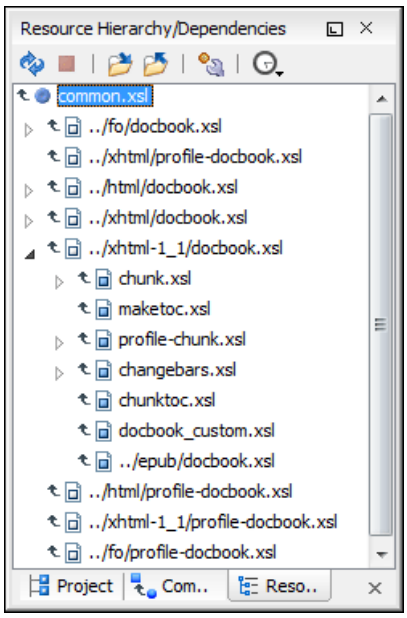


Figure 111: Resource Hierarchy/Dependencies View - Dependencies for `common.xsl`

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

- - Refreshes the hierarchy/dependencies structure.
- - Stop the hierarchy/dependencies computing.
- - Allows you to choose a schema to compute the hierarchy structure.
- - Allows you to choose a schema to compute the dependencies structure.
- - Allows you to configure a scope to compute the dependencies structure.
- - Allows you to repeat a previous dependencies computation.

The following actions are available in the contextual menu:

- **Open** - Opens the schema. Alternatively, you can open the schema by a double-click on the Hierarchy/Dependencies structure.
- **Copy location** - Copies the location of the schema.
- **Show Resource Hierarchy** - Shows the hierarchy for the selected schema.
- **Show Resource Dependencies** - Shows the dependencies for the selected schema.
- **Expand All** - Expands all the children of the selected schema from the Hierarchy/Dependencies structure.
- **Collapse All** - Collapses all the children of the selected schema from the Hierarchy/Dependencies structure.

Component Dependencies View

The Component Dependencies view allows you to see the dependencies for a selected XSLT component. You can open the view from **Window** > **Show View** > **Component Dependencies** .

If you want to see the dependencies of an XSLT component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named components (templates, variables, parameters, attribute sets, keys, etc).

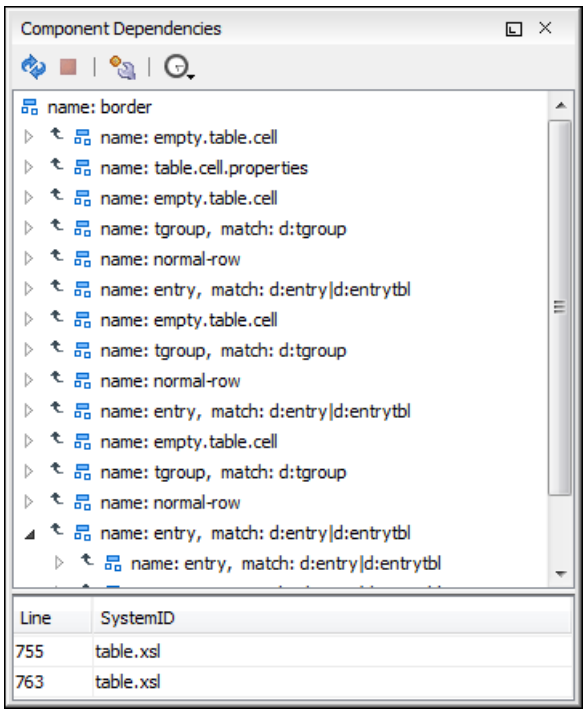


Figure 112: Component Dependencies View - Hierarchy for table.xsl

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

The Quick Assist action set was designed to help you improve the development work flow by offering quicker access to the most commonly used actions.

It is activated automatically when the cursor is positioned inside a component name. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press **Alt + 1** keys (**Meta + Alt + 1** on Mac OS X).

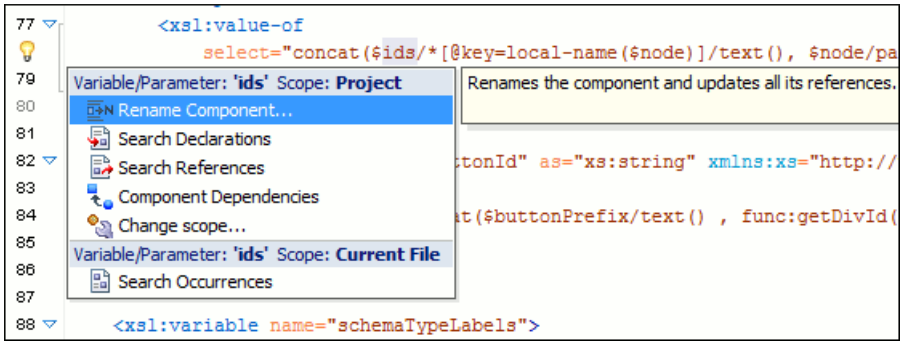


Figure 113: XSLT Quick Assist Support

The quick assist support offers direct access to the following actions:

- **Rename Component** - Renames the component and all its dependencies;
- **Search Declaration** - 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;
- **Search Occurrences** - Searches all occurrences of the file within the current scope.

Linking Between Development and Authoring

The Author page is available for the XSLT editor presenting the stylesheets in a nice visual rendering. For more details, see *The XSLT Document Type*.

Editing XQuery Documents

This section explains the features of the XQuery editor and how they should be used.

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 a quick access to a component by knowing its name. It is opened from menu **Window** > **Show View** > **Outline** .

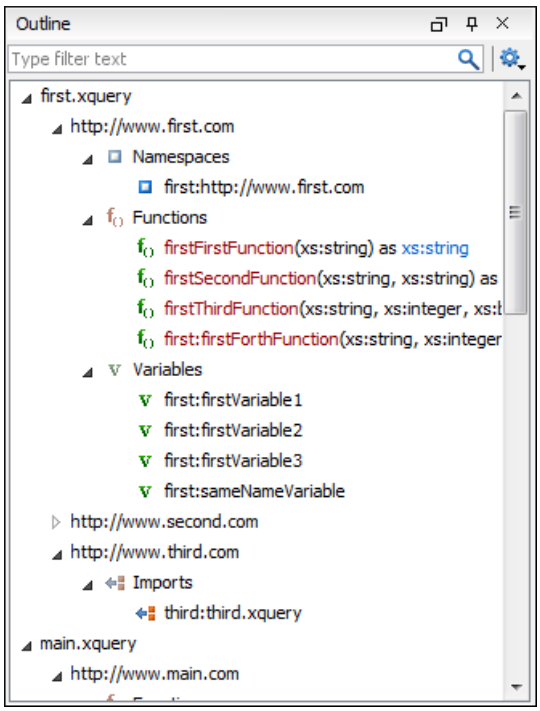


Figure 114: XQuery Outline View

The following actions are available in the **Settings** menu on the Outline view's toolbar:

- **Selection update on caret move** - Allows a synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret's moves or the changes 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 sort alphabetically the XQuery components.
- **Show all components** - Displays all components that were collected starting from the current file. This option is set by default.
- **Show local components** - Displays 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 the first presented 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 bottom 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 searched 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.

Figure 115: Folding in XQuery Documents

There is available the action **Go to Matching Bracket**<u>**Ctrl+Shift+G**</u> 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 **Tools** > **Generate Documentation** > **XQuery Documentation...** . . It can be also opened from the **Project Tree**'s **contextual menu** > **Generate Documentation** > **XQuery Documentation...** . . The dialog enables the user to configure a set of parameters of the process of generating the HTML documentation. The parameters are:

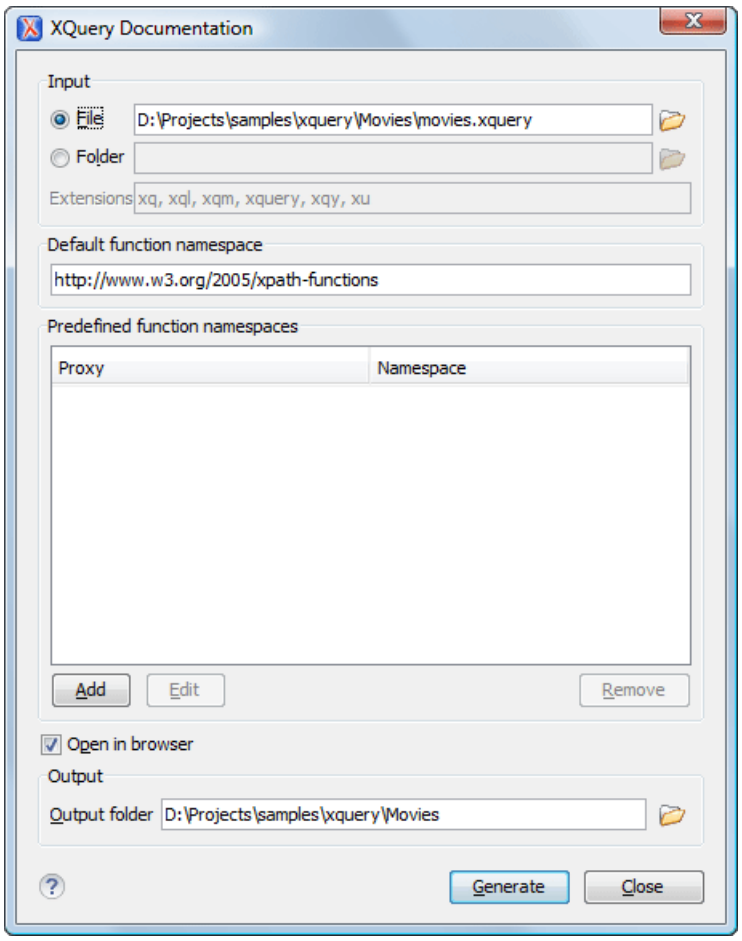


Figure 116: The XQuery Documentation Dialog

- **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** - When checked, the generated documentation is opened in an external browser.
- **Output** - Allows the user to specify where the generated documentation is saved on disk.

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 includes a built-in CSS validator integrated with the general validation support, bringing the *usual validation features* to CSS stylesheets.

When the current editor is a CSS type one, the **Validate** toolbar provides a **Validation options** button for quick access to the *CSS validator options* in the Oxygen user preferences.

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 - Space)** shortcut and it is context-sensitive when invoked for the value of a property.

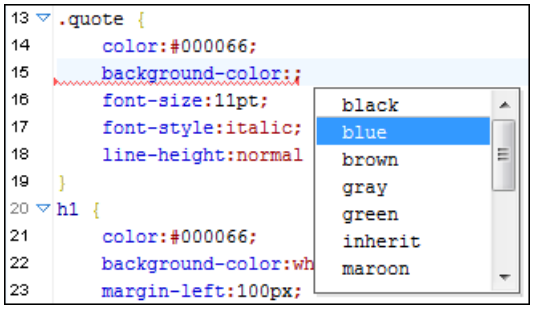


Figure 117: Content Completion in CSS Stylesheets

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.

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.

The selection in the **Outline** view can be synchronized with the caret moves or the changes made in the stylesheet document. When selecting an entry from the **Outline** view, the corresponding import or selector is highlighted in the CSS editor.

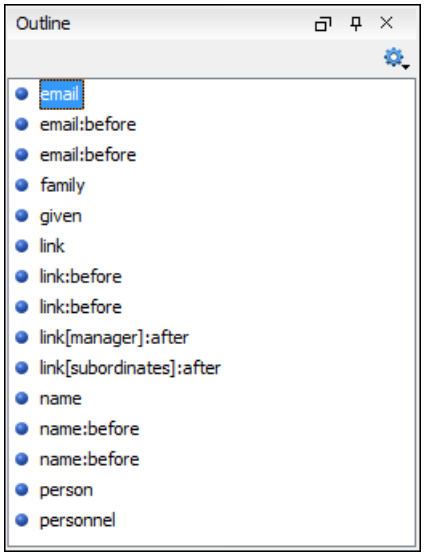


Figure 118: CSS Outline View

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.

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 split actions, the folding actions, the edit actions* and a part of *the source actions* (only the actions **To lower case**, **To upper case**, **Capitalize lines**) are available.

Editing 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 version 12.2.

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 element `xslt` is the parent of element `input`, the content completion offers XSLT elements.
- If the value of the `port` attribute is `schema` and element `validate-with-relax-ng` is the parent of element `input`, the content completion offers RELAX NG schema elements.
- If the value of the `port` attribute is `schema` and element `validate-with-xml-schema` is the parent of element `input`, the content completion offers XML Schema schema elements.
- If the value of the `port` attribute is `schema` and element `validate-with-schematron` is the parent of element `input`, the content completion offers either ISO Schematron elements or Schematron 1.5 schema elements.
- If the above cases do not apply, then the content completion window offers elements from all the schemas from the above cases.

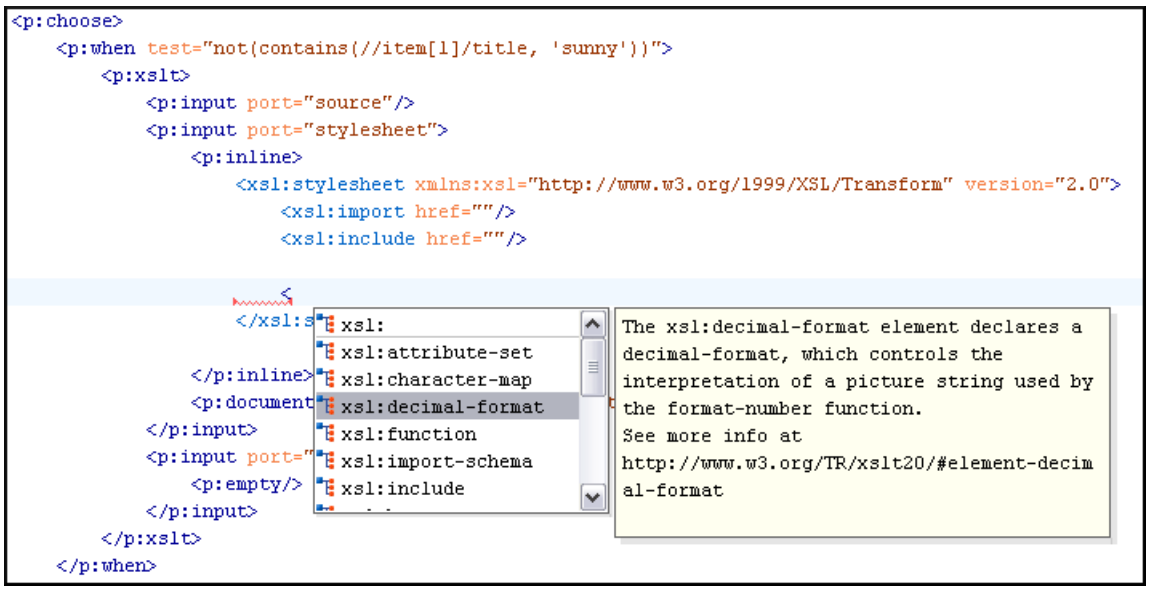


Figure 119: XProc Content Completion

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 (as opposed to a cryptic error code) which will be provided to the user if an error is encountered during validation stage.

Oxygen uses for validation the Skeleton XSLT processor and conforms with ISO Schematron or Schematron 1.5. It allows you to validate XML documents against Schematron schema or against combined RELAX NG / W3C XML Schema and Schematron.

Oxygen assists you in editing Schematron documents by providing schema-based content completion and syntax coloring. A basic Schematron template is available in the **New Document** wizard.

Any time you can validate the content using the **Validate** action. Another way to validate schemas is to check them against their own Schematron schema rules using **External validation** action.

Combined RELAX NG / W3C XML Schemas and Schematron Schema

Schematron rules can be embedded into W3C Schema through annotation (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 accepts such documents as Schematron validation schemas and it is able to extract and use the embedded rules. To validate a document with both RELAX NG schema and its embedded Schematron rules, you need two persistence associations like in the following example:

```
<?oxygen RNGSchema="percent.rng">
<?oxygen SCHSchema="percent.rng">
```

The second association validates your document with Schematron rules extracted from the RELAX NG Schema. Similarly you can specify as Schematron Schema a W3C XML Schema having the Schematron rules embedded:

```
<?oxygen SCHSchema="percent.xsd">
```

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:

```
<?oxygen SCHSchema="percent.sch"?>
```

The possible errors which might occur during the validation process are presented in the **Errors** panel at the bottom area of the Oxygen window. Each error is flagged with a severity level, which Errors are flagged with a security level, which can be one of *warning*, *error*, *fatal* or *info*.

To set a severity level, Oxygen 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 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 sets the security level to *error*.

SVG Documents

SVG is a platform for two-dimensional graphics. It has two parts: an XML-based file format and a programming API for graphical applications. Just to enumerate some of the key features: shapes, text, and embedded raster graphics with many painting styles, scripting through languages such as ECMAScript and support for animation.

SVG is a vendor-neutral open standard that has important industry support. Companies like Adobe, Apple, IBM, and others have contributed to the W3C specification. Many documentation frameworks, including DocBook, have support for SVG by defining the graphics directly in the document.

Oxygen XML Editor adds SVG support by using the *Batik* package, an open source project developed by the Apache Software foundation. *Oxygen's default XML catalog* solves the SVG DTD.

👉 **Tip:** To render SVG images which use Java scripting, copy the `js.jar` library from the Batik distribution into the Oxygen `lib` folder and restart the application.

👉 **Tip:** There are many navigation shortcuts which can be used for navigation in the SVG Viewer like:

- The arrow keys or **(Shift + Left Click)** move the image.
- **(Ctrl + Right Click)** rotates the image.
- **(Ctrl + I)** and **(Ctrl + O)** or **(Ctrl + Left Click)** to zoom in or out.
- **(Ctrl + T)** to reset the transform.

The Standalone SVG Viewer

You can use the action **Tools** > **SVG Viewer ...** to browse and open any SVG file having the `.svg` or `.svgz` extension. If the file is included in the current project, then you can open it by right-clicking on it and selecting **Open with** > **SVG Viewer** . The following actions are available in a contextual menu:

- **Zoom in** - Zooms in the image by a factor of 2. The action is also available on **Mouse Wheel Up**;
- **Zoom out** - Zooms out the image by a factor of 2. The action is also available on **Mouse Wheel Down**;
- **Rotate** - Rotates the image 90 degrees clockwise;
- **Refresh** - Refreshes the image, by reloading the SVG file.

Figure 120: SVG Viewer

The Preview Result Panel

This panel can render the result of an *XSL transformation* that generates SVG documents.

Figure 121: Integrated SVG Viewer

The basic use-case of Oxygen consists in the development of the XSL stylesheets capable of producing rich SVG graphics. For example, you have an XML document describing the evolution of a parameter over time and you create

a graphic from it. You can start with a static SVG, written directly in Oxygen or exported from a graphics tool like the Adobe suite. Extract then the parts that are dependent of the data from the XML document and create the XSL templates. Select the option **Show as SVG** in *the dialog for configuring the XSLT transformation scenario*. When you run the transformation, the SVG result is displayed in the SVG result panel.

Integrating External Tools

Sometimes an external tool which can be launched from the command line and which is different than a *FO processor* is needed. Oxygen offers you the option of integrating such a tool by specifying just the command line for starting the executable file and its working directory. To integrate such a tool, *go to* ***Options*** > ***Preferences*** > ***External Tools***

If the external tool is applied on one of the files opened in Oxygen, *enable the option* for saving all edited files automatically when an external tool is applied.

External tools can be launched from the **External tools** toolbar or from the submenu **Tools** > **External tools** . While the action is running its icon is a stop icon:  . When the tool has finished running, it changes the icon back to the original run icon:  . Please note that even though you can stop the external tool by invoking the action again while it is running, that doesn't mean you can also stop the processes spawned by that external tool. This is especially a limiting factor when running a batch file as the batch will be stopped but without actually stopping the processes that the batch was running at that time.

Integrating the Ant Tool

As example let us integrate *the Ant build tool* in Oxygen:

- *Download* and *install* *Ant* on your computer;
- Test your *Ant* installation from the command-line interface in the directory where you want to use *Ant* from Oxygen, for example run the `clean` target of your `build.xml` file `C:\projects\XMLproject\build.xml`:

  ```
  ant clean
  ```

- *Go to* ***Options*** > ***Preferences*** > ***External Tools*** ;
- Create a new external tool entry with the name **Ant tool**, the working directory `C:\projects\XMLproject` and the command line `"C:\projects\XMLproject\ant.bat" clean` obtained by browsing to the `ant.bat` file from directory `C:\projects\XMLproject`;
- Run the tool from **Tools** > **External Tools** > **Ant tool** . You can see the output in the **Command** results panel:

  ```
  Started: "C:\projects\XMLproject\ant.bat" clean
  Buildfile: build.xml

  clean:
  [echo] Delete output files.
  [delete] Deleting 5 files from C:\projects\XMLproject

  BUILD SUCCESSFUL
  Total time: 1 second
  ```

Editing Very Large Documents

For editing very large documents (file size up to 300 MB), a special memory optimization is implemented on loading such a file so that the total memory allocated for the application is not exceeded. The minimum file size that enables this large file optimization can be *configured with the option* ***Optimize loading in the Text page for files over (MB)*** available from menu **Options** > **Preferences** > **Editor** > **Open/Save** .

A temporary buffer file is created on disk so you have to make sure that the available free disk space is at least double the size of the large file that you want to edit. For example Oxygen can load a 200-MB file using a minimum memory setting of 512 MB and at least 400-MB free disk space.

The increase of the maximum size of editable files comes with the following restrictions:

- A file larger than the value of the above option is edited only in Text mode.
- The *automatic validation* is not available when editing a very large file.
- The XPath filter is disabled in *the Find/Replace dialog*.
- The bidirectional Unicode support (right-to-left writing) is disabled.
- *The option* ***Format and indent the document on open*** is disabled for non-XML documents. For XML documents, it is done optimizing the memory usage but without respecting the options set in *the* ***Format*** *preferences page*.
- Less precise localizations for the results of an *XPath expression*.

Insufficient Memory

If the application displays an *out of memory* (**OutOfMemoryError**) error when you try to edit very large files, this means that the memory allocated to the application is insufficient. Apply one or more steps from the following list to avoid the error:

- Use the *-Xmx parameter* to adjust the maximum memory available to the application at startup.
- Make sure that you close other files before opening the large file.
- The large file is opened in *Text editing mode* because it uses less memory than other editing modes. You can set the default editing mode *in the Preferences dialog*.
- If the file is too large for the editor to handle, you can *open it in* ***Large File Viewer***.

Large File Viewer

XML files tend to become larger and larger mostly because they are frequently used as a format for database export or for porting between different database formats. Traditional XML text editors simply cannot handle opening these huge export files, some having sizes exceeding one gigabyte, because all the file content must be loaded in memory before the user can actually view it.

The best performance of the viewer is obtained for encodings that use a fixed number of bytes per character, like UTF-16 or ASCII. The performance for UTF-8 is very good for documents that use mostly characters of the European languages. For the same encoding, the rendering performance is higher for files consisting of long lines (up to few thousands characters) and may degrade for short lines. In fact, the maximum size of a file that can be rendered in the Large File Viewer decreases when the total number of the text lines of the file increases. Trying to open a very large file, for example a file of 4 GB with a very high number of short lines (100 or 200 characters per line) may produce an *out of memory* error (**OutOfMemoryError**) which would require either increasing the Java heap memory with the -Xmx startup parameter or decreasing the total number of lines in the file.

The powerful **Large File Viewer** is available from the **Tools** menu or as a standalone application. You can also right click a file in your project and choose to open it with the viewer. It uses an efficient structure for indexing the opened document. No information from the file is stored in the main memory, just a list of indexes in the file. In this way the viewer can open very large files, up to 10 gigabytes. If the opened file is XML, the encoding used to display the text is detected from the XML prolog of the file. For other file types, the encoding is taken from the Oxygen options. See *Encoding for non XML files*.

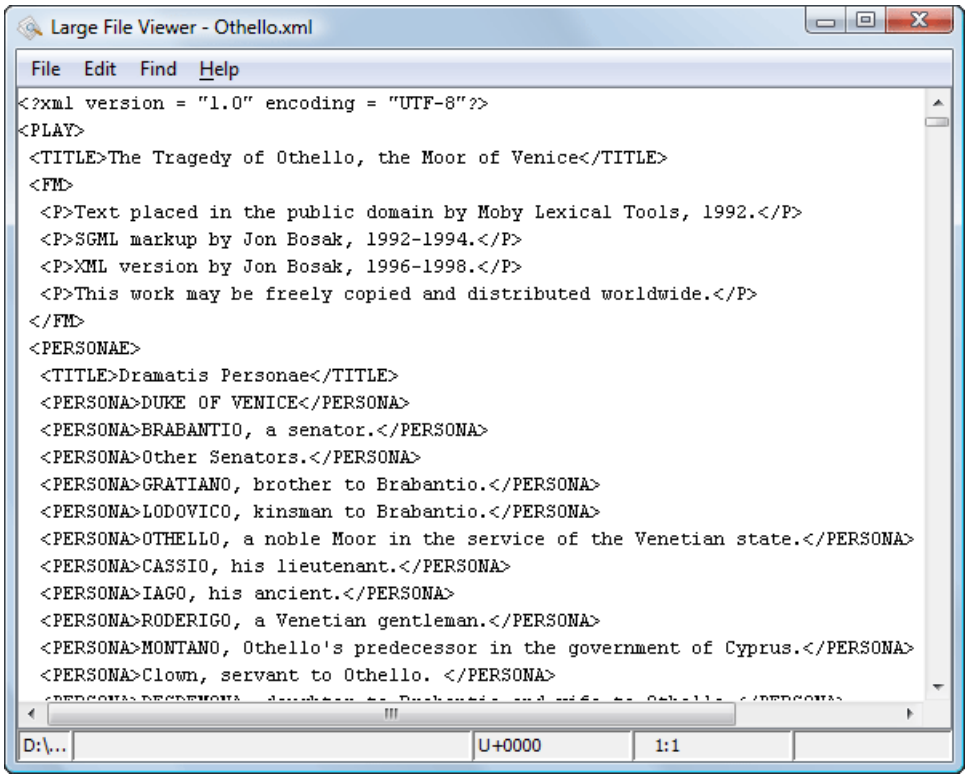


Figure 122: The Large File Viewer

Large File Viewer components:

- The menu bar provides menu driven access to all the features and functions available in **Large File Viewer**.
  - **File** > **Open** opens files in the viewer (also available in the contextual pop-up menu).
  - **File** > **Close** closes the viewer.
  - **Edit** > **Copy** copies the selected text to clipboard (also available in the contextual pop-up menu).
  - **Find** > **Find** opens a reduced **Find** dialog providing some basic search options like:
    - **Case sensitive** - When checked, operations are case-sensitive.
    - **Regular Expression** - When checked, allows using any regular expression in *PERL* syntax.
    - **Wrap around** - Continues the find from the start (end) of the document after reaching the end (start) if the search is in forward (backward) direction.
  - **Help** > **Help** provides access to this User Manual.
- The status bar provides information about the current opened file path, the Unicode representation of the character at caret position and the line and column in the opened document where the caret is located.

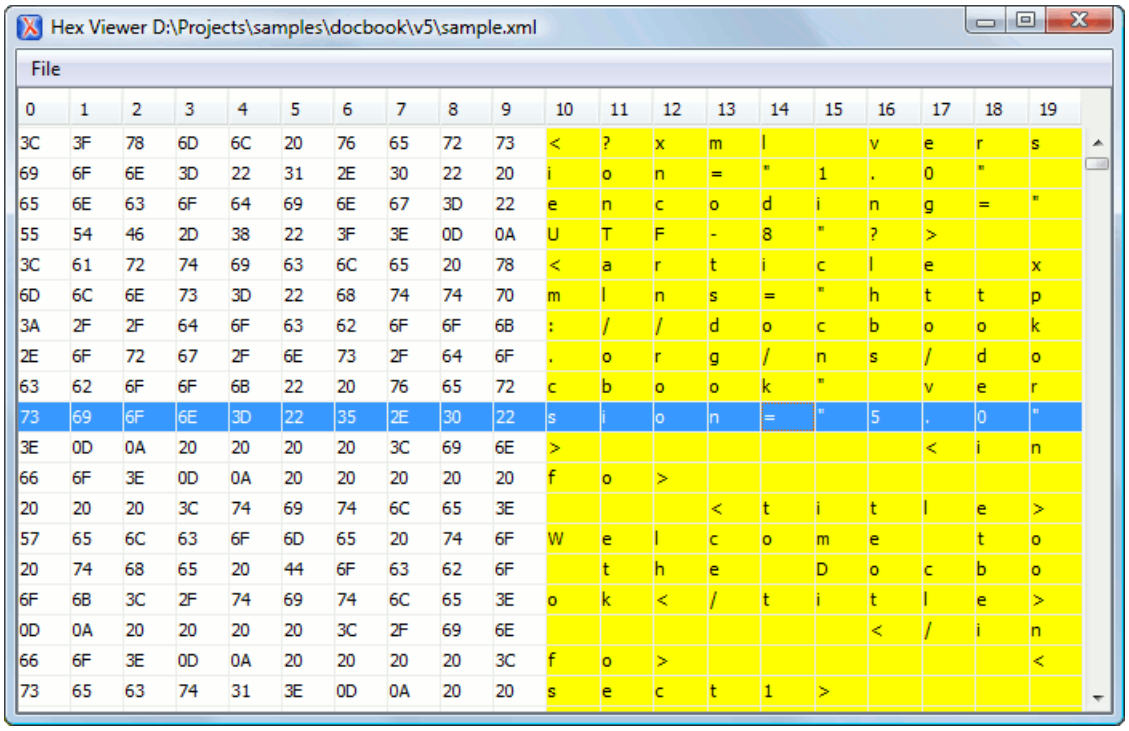
⚠ **Attention:** For faster computation the **Large File Viewer** uses a fixed font (plain, monospace font of size 12) to display characters. The font is *not* configurable from the Oxygen **Preferences** page.

👉 **Tip:** The best performance of the viewer is accomplished for encodings that use a fixed number of bytes per character, like UTF-16 or ASCII. The performance for UTF-8 is very good for documents that use mostly characters

of the European languages. For the same encoding the rendering performance is high for files consisting of short lines (up to a few thousand characters) and may degrade for long lines.

Hex Viewer

When the Unicode characters that are visible in a text viewer or editor are not enough and you need to see the byte values of each character of a document, you can start the hex viewer that is available on the **Tools** menu. It has two panels: the characters are rendered in the right panel and the bytes of each character are displayed in the left panel. There is a 1:1 correspondence between the characters and their byte representation: the byte representation of a character is displayed in the same matrix position of the left panel as the character in the matrix of the right panel.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3C	3F	78	6D	6C	20	76	65	72	73	<	?	x	m	l		v	e	r	s
69	6F	6E	3D	22	31	2E	30	22	20	i	o	n	=	"	1	.	0	"	
65	6E	63	6F	64	69	6E	67	3D	22	e	n	c	o	d	i	n	g	=	"
55	54	46	2D	38	22	3F	3E	0D	0A	U	T	F	-	8	"	?	>		
3C	61	72	74	69	63	6C	65	20	78	<	a	r	t	i	c	l	e		x
6D	6C	6E	73	3D	22	68	74	74	70	m	l	n	s	=	"	h	t	t	p
3A	2F	2F	64	6F	63	62	6F	6F	6B	:	/	/	d	o	c	b	o	o	k
2E	6F	72	67	2F	6E	73	2F	64	6F	.	o	r	g	/	n	s	/	d	o
63	62	6F	6F	6B	22	20	76	65	72	c	b	o	o	k	"		v	e	r
73	69	6F	6E	3D	22	35	2E	30	22	s	i	o	n	=	"	5	.	0	"
3E	0D	0A	20	20	20	20	3C	69	6E	>							<	i	n
66	6F	3E	0D	0A	20	20	20	20	20	f	o	>							
20	20	20	3C	74	69	74	6C	65	3E				<	t	i	t	l	e	>
57	65	6C	63	6F	6D	65	20	74	6F	W	e	l	c	o	m	e		t	o
20	74	68	65	20	44	6F	63	62	6F		t	h	e		D	o	c	b	o
6F	6B	3C	2F	74	69	74	6C	65	3E	o	k	<	/	t	i	t	l	e	>
0D	0A	20	20	20	20	3C	2F	69	6E							<	/	i	n
66	6F	3E	0D	0A	20	20	20	20	3C	f	o	>							<
73	65	63	74	31	3E	0D	0A	20	20	s	e	c	t	1	>				

Figure 123: Hex Viewer

Scratch Buffer

A handy addition to the document editing is the **Scratch Buffer** view used for storing fragments of arbitrary text during the editing process. It can be used to drop bits of paragraphs (including arbitrary XML markup fragments) while rearranging and editing the document and also to drag and drop fragments of text from the scratch buffer to the editor panel. The **Scratch Buffer** is basically a text area offering XML syntax highlight. The view contextual menu contains basic edit actions like **Cut**, **Copy**, and **Paste**.

Changing the User Interface Language

Oxygen comes with a user interface available in English, French, German, Japanese, Dutch, and Italian. If you want to use Oxygen in other language you have to translate all the messages and labels available in the user interface (menu

action names, button names, check box texts, view titles, error messages, status bar messages, etc.) and provide a text file with all the translated messages in the form of a Java properties file. Such a file contains *message key - translated message* pairs displayed in the user interface. To install the new set of translated messages, copy this file into the `[Oxygen-install-folder]/lib` folder, restart Oxygen and *set the new language in the Oxygen preferences.* You can get the keys of all the messages that must be translated from the properties file containing the English translation. To get this file, contact us at support@oxygenxml.com.

Handling Read-Only Files

If a file marked as read-only is opened in Oxygen you can by default perform modifications to it. This behavior is controlled by the ***Can edit read only files*** option. When attempting to save such files you will be prompted to save them to another location.

You can check out the read-only state of the file by looking in the *Properties view*. If you modify the file properties from the operating system and the file becomes writable, you are able to modify it on the spot without having to reopen it.

The read-only state is marked with a lock decoration which appears in the editor tab and specified in the tooltip for a certain tab.

Editing Documents with Long Lines

The documents containing long lines can affect performance when opened in the text editor. If you choose to present the document with line wrap, some features are affected:

- The editor uses the `Monospaced` font.
- You cannot set font styles from **Options** > **Preferences** > **Editor** > **Colors** .
- Automatic validation is disabled.
- Automatic spell checking is disabled.
- **XPath** field is disabled in the **Find/Replace** dialog.
- Less precise localization for executed XPaths. The XPath executions use SAX sources for smaller memory footprint. We recommend using XPath 2.0 instead of XPath 1.0 because it has increased speed and a smaller memory footprint. Running an XPath expression requires additional memory about 2 or 3 times the size of the document on disk.

The last two restrictions are valid only for XML documents.

Chapter 5

Authoring in the Tagless Editor

Topics:

- *Authoring XML Documents Without the XML Tags*
- *General Author Presentation*

This chapter presents the WYSIWYG like editor targeted for content authors, also called Author editor.

Authoring XML Documents Without the XML Tags

Once the structure of the XML document and the required restrictions on the elements and attributes are fixed with an XML schema the editing of the document is 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. The Author mode is activated by pressing the Author button at the bottom of the editing area where the mode switches of the XML editor are available: Text, Grid, and Author. 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.

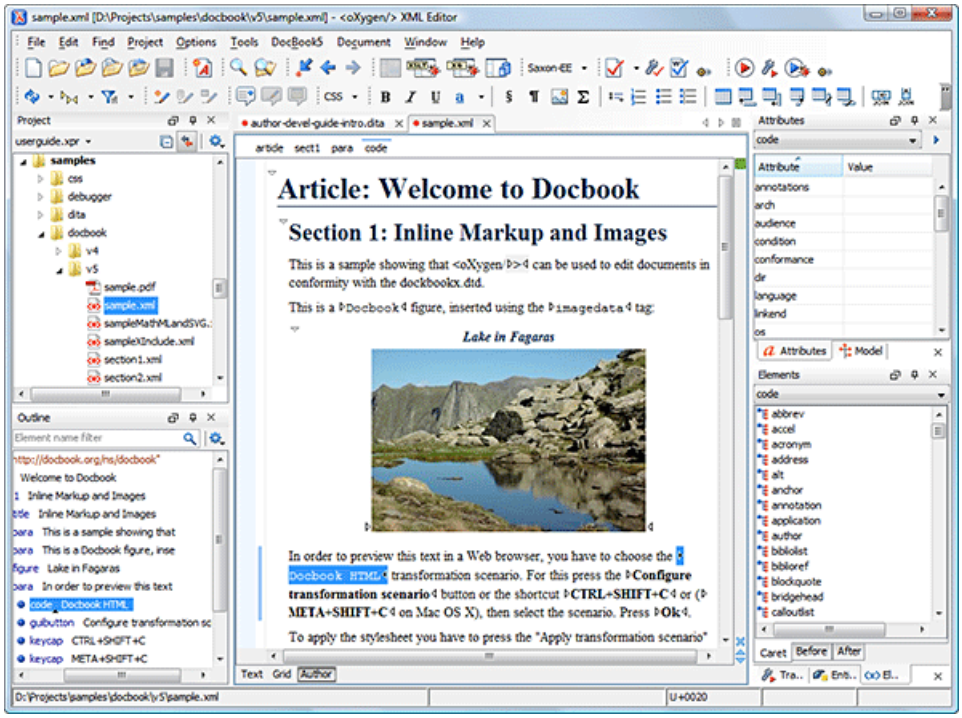


Figure 124: oXygen Author Editor

The tagless rendering of the 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. Also some CSS 3 features like namespaces and custom extensions of the CSS specification are supported.

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 as it is a plain text file with a simple syntax.

The association of such a stylesheet with an XML document is also straightforward: an `xml-stylesheet` XML processing instruction with the attribute `type="text/css"` must be inserted at the beginning of the XML document. If it is an XHTML document, that is the root element is an `html` element, 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 CSS specification*.

There are two main types of users of the Author mode: *developers* and *content authors*. A *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.

General Author Presentation

A content author edits the content of XML documents in tagless 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.

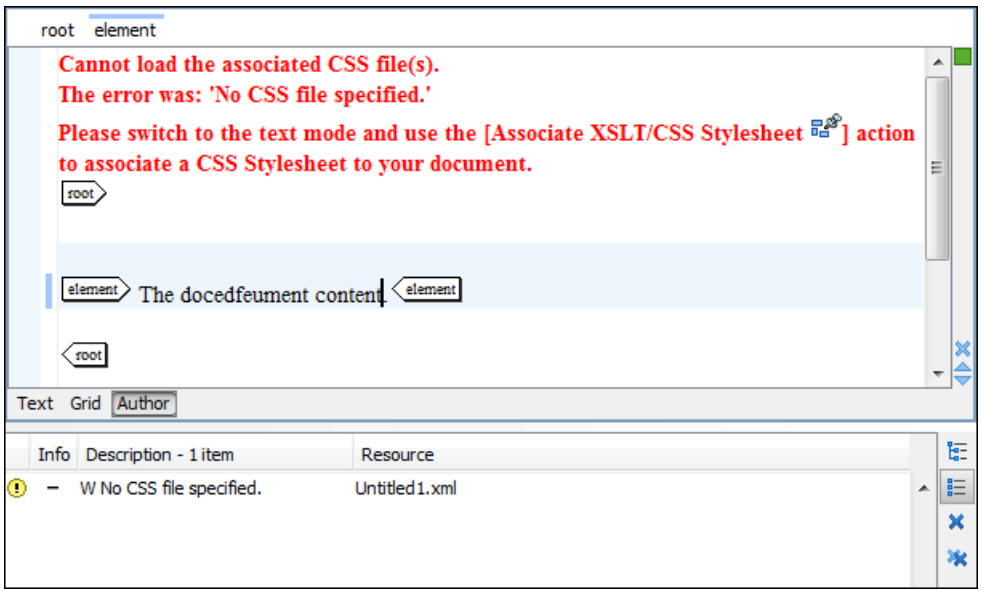


Figure 125: Document with no CSS association default rendering

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*

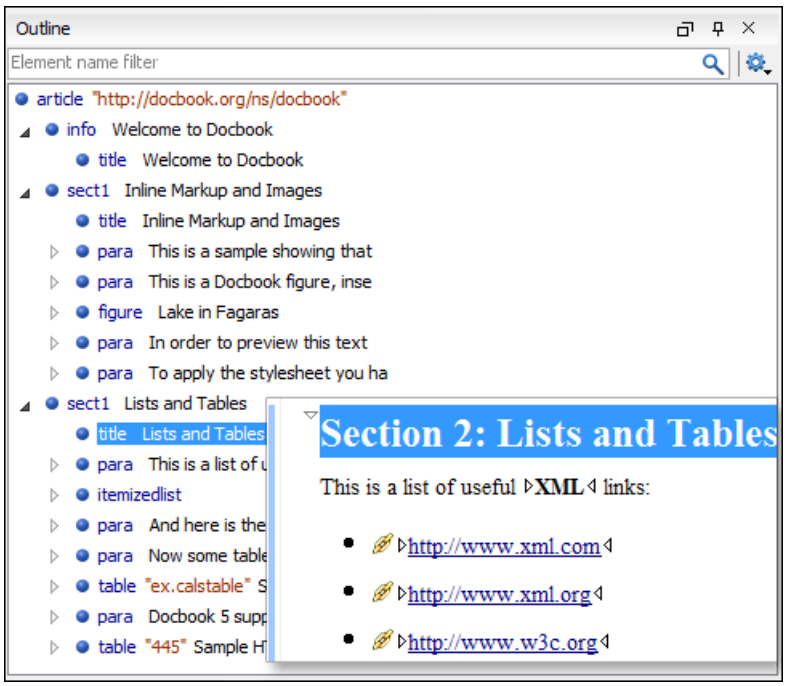


Figure 126: The Outline View

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. That makes easier for the user to be aware of the document structure and the way 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 *Expand more* and *Collapse all* items of the popup menu available on the Outline tree enlarge or reduce the set of nodes of the edited document currently visible in the view. The tree expansion action is a faster alternative to mouse clicks on the plus signs of the tree when one wants to access quickly a node deeply nested in the hierarchy of document nodes. When a large number of nodes become expanded and the document structure is not clear any more, the collapsing action clears the view quickly by reducing the depth of the expanded nodes to only one child of the currently selected node.

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. An easy-to-spot exclamation mark sign is used as element icon, a red underline decorates the element name and value and a tooltip provides more information about the nature of the error.

Modification Follow-up

When editing, the Outline view dynamically follows the modifications introduced by the user, showing in the middle of the panel the node which is currently being modified. This gives the user better insight on location where in the document one is positioned and how the structure of the document is affected by one's modifications.

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)** key after dragging, there will be performed a copy operation instead of a move one.

The drag and drop action in the **Outline** view can be *disabled and enabled from the Preferences dialog.*

Tip: You can select and drag multiple nodes in the Author Outline tree.

Outline Filters

The following actions are available in the **Settings** menu on the Outline view's toolbar:

- **Flat presentation mode of the filtered results** - when active, the application flattens the filtered result elements to a single level.
- **Show comments and processing instructions** - show/hide comments and processing instructions in the **Outline** view.
- **Show 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.
- 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 More / 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 will update the list of the allowed elements in **Before** and **After** tabs.

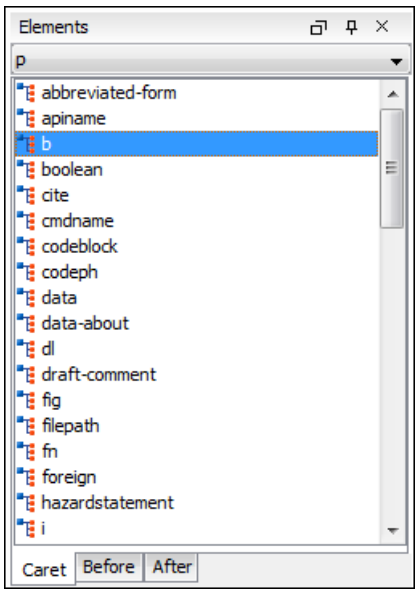


Figure 127: The Elements View

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 will insert 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 will insert 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 will insert 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 possible attributes of the current element allowed by the schema of the document. It allows you to insert attributes in the current element or change the value of the attributes already used in the element. The already present attributes are painted with a bold font. Default values are painted with gray color. The **Attributes** view uses a red color to highlight invalid attributes and values.

Clicking the **Value** column of a table row will start editing the value of the attribute from the selected row. If the possible values of the attribute are specified as list in the schema associated with the edited document the **Value** column works as a combo box where you can select one of the possible values to be inserted in the document. The attributes table is sortable by clicking the column names. Thus the table's contents can be sorted in ascending order, in descending order or in a custom order, where the used attributes are placed 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.

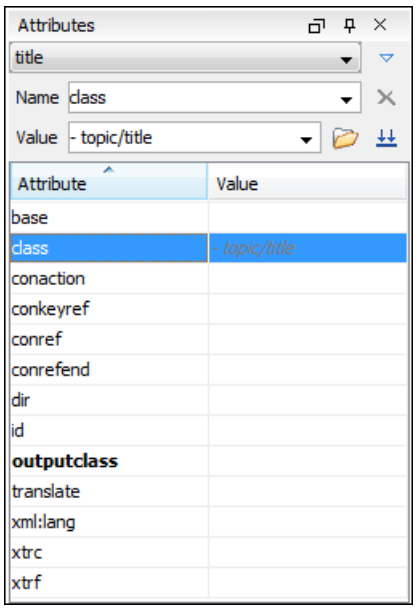


Figure 128: The Attributes View

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 contextual menu of the view allows you to insert a new element (**Add** action) or delete an existing one (**Delete** action). Delete action can be invoked on a selected table entry by pressing **(Del)** or **(Backspace)**.

The attributes of an element can be edited also in place in the editor panel by pressing the shortcut **(Alt + Enter)** which pops up a small window with the same content of the **Attributes** view. In the initial form of the popup, only the two text fields **Name** and **Value** are displayed, the list of all the possible attributes is collapsed.

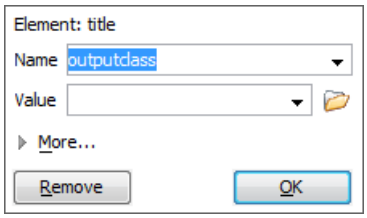


Figure 129: Edit attributes in place

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

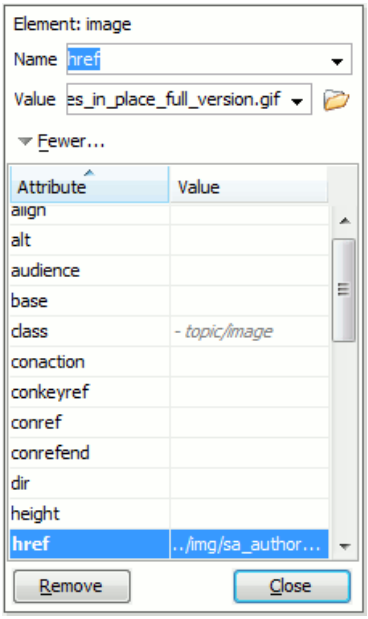


Figure 130: Edit attributes in place - full version

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.

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.

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.

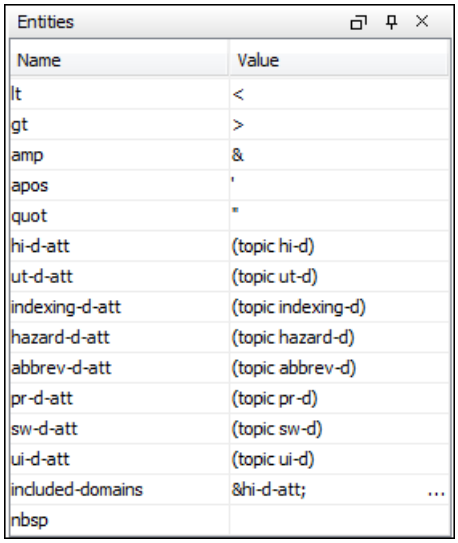


Figure 131: The Entities View

The Author Editor

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

Navigating the Document Content

Fast navigating the document content can be done using the **(Tab)**/**(Shift + Tab)** for advancing forward / backwards. The caret is moved to the next / previous editable position. To navigate one word forward or backwards, use **Ctrl + Right Arrow**, and **Ctrl + Left Arrow**, respectively. Entities and hidden elements are skipped.

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

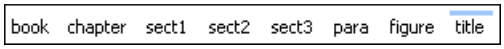


Figure 132: The breadcrumb in Editor view

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 locations of selected text are stored in an internal list which allows navigating between them with the buttons **Ctrl+Alt+[** **Back** and **Ctrl+Alt+]** **Forward** that are available on the toolbar **Navigation**.

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. 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.

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.

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 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+Home)** and **(Ctrl+End)**, respectively.

Displaying the Markup

In Author view, the amount of displayed markup can be controlled using the following dedicated actions:

- **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** - None of the tags is displayed. This is the most compact mode.

The default tags display mode can be configured in the *Author options page*. 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.

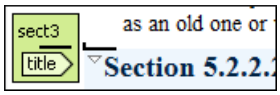


Figure 133: Before first block

- The caret is positioned between two block elements.

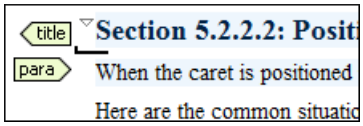


Figure 134: Between two block elements

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

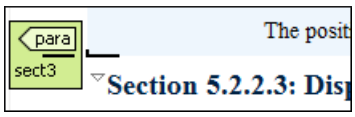


Figure 135: After last block

- The caret is positioned inside a node.

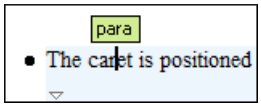


Figure 136: Inside a node

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

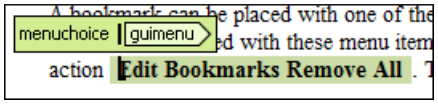


Figure 137: Before an inline element

- The caret is positioned between two inline elements.

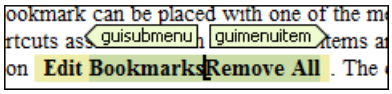


Figure 138: Between two inline elements

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

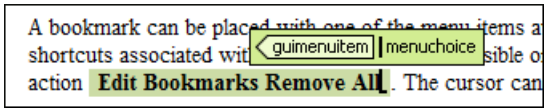


Figure 139: After an inline element

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 options page*. The referred resources are loaded and displayed inside the element or entity that refers them, however the displayed content cannot be modified directly.

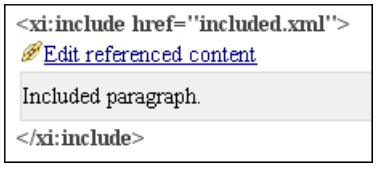


Figure 140: XInclude reference

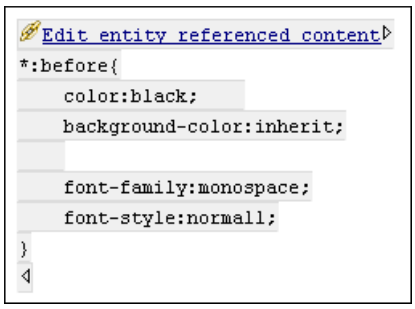


Figure 141: External entity reference

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 on the link (marked with the icon) which is displayed before the referred content. 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 Author;
- on demand, by using the *Refresh references action*. Useful when the referred content is modified outside the Oxygen XML Author scope.

Finding and Replacing Text

The ***Find / Replace** dialog* can be used in the Author page in the same way as in the Text page. However, there are some features which are disabled:

- search in XPath
- search in selection
- search in tags

These limitations can be compensated by using the ***Find All Elements** dialog*.

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).

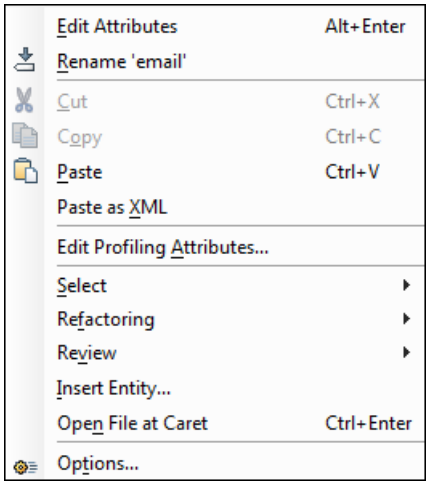


Figure 142: Contextual menu

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.

- **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.
  - **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 tag.
  - **Surround with '<Tag name>'** - Selected text in the editor is marked with start and end tags of 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.
- **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 name under the current position of the caret in the current document. 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 section *Predefined document types.*

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 + Space)** (on *Mac OS X* the shortcut is **(Meta + 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*.

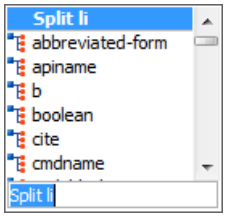


Figure 143: Content completion window

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 considered by the associated schema as valid proposals in the current context. This can be changed by unchecking the **Allow only insertion of valid elements and attributes** check box from the *Schema aware preferences page*.

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 if the word at cursor position is a prefix of such a short name. If there is no prefix at cursor position, that is the character at the left of cursor is a whitespace, all the code templates are listed.

Oxygen comes with a lot of predefined code templates but you can *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 can use the Content Completion on request shortcut key (usually CTRL-SPACE) or the Code Templates on request shortcut key (CTRL-SHIFT-SPACE). 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 following variables can appear in a code template:

- *${caret}* - The caret position after inserting the code template.
- *${selection}* - The position of the current selection in the inserted template.
- *${env(ENV_VAR_NAME)}* - The value of the environment variable *ENV_VAR_NAME*.
- *${system(var.name)}* - The value of the system variable *var.name*.
- *${date(yyyy-MM-dd)}* - A date in the format: 4 digits for year, 2 digits for month, 2 digits for day.

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:

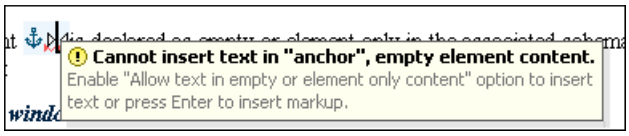


Figure 144: Editing in empty element warning

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.

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, etc. 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*.

More details about setting up a custom copy/paste handler are *available in the Author Developer Guide*.

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 - NumPad+)** or **(Ctrl - +)** or **(Ctrl - mouse wheel)** - Increases font size.
- **(Ctrl - NumPad-)** or **(Ctrl - -)** or **(Ctrl - mouse wheel)** - Decreases font size.
- **(Ctrl - NumPad0)** or **(Ctrl - 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 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. If you cut and paste content, the moved element retains its ID.

Table Layout and Resizing

The support for editing data in tabular form can manage table width and column width specifications from the source document. The specified widths will be considered when rendering the tables and when visually resizing them using mouse drag gestures. These specifications are supported both in fixed and proportional dimensions. The predefined frameworks (DITA, DocBook and XHTML) already implement support for this feature. The layout of the tables from these types of documents takes into account the table width and the column width specifications particular to them. The tables and columns widths can be visually adjusted by dragging with the mouse their edges and the modifications will be committed back into the source document.

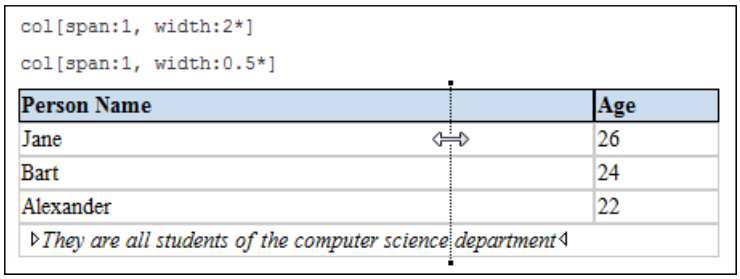
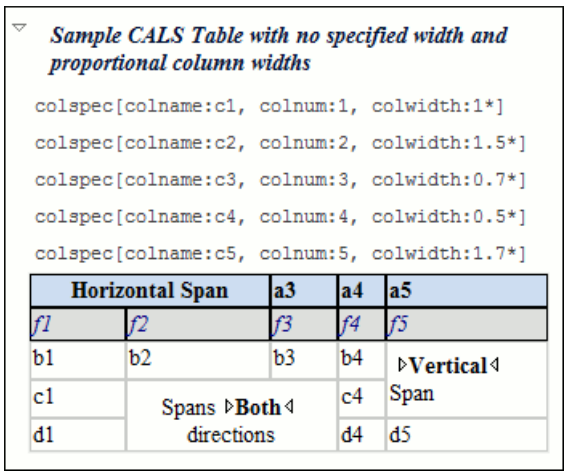


Figure 145: Resizing a column in Oxygen Author editor

DocBook Table Layout

The DocBook table layout supports two models: CALS and HTML.

In the CALS model column widths can be specified by using the `colwidth` attribute of the associated `colspec` element. The values can be fixed or proportional.



Sample CALS Table with no specified width and proportional column widths

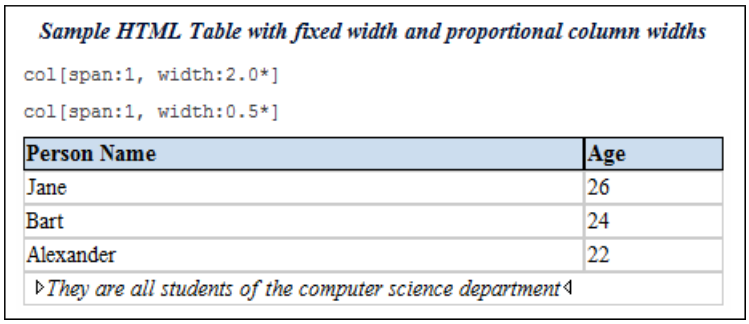
```
colspec[colname:c1, colnum:1, colwidth:1*]
colspec[colname:c2, colnum:2, colwidth:1.5*]
colspec[colname:c3, colnum:3, colwidth:0.7*]
colspec[colname:c4, colnum:4, colwidth:0.5*]
colspec[colname:c5, colnum:5, colwidth:1.7*]
```

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 146: CALS table

XHTML Table Layout

The HTML table model accepts both table and column widths by using the width attribute of the table element and the col element associated with each column. The values can be represented in fixed units, proportional units or percentages.



Sample HTML Table with fixed width and proportional column widths

```
col[span:1, width:2.0*]
col[span:1, width:0.5*]
```

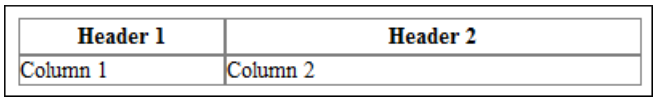
Person Name	Age
Jane	26
Bart	24
Alexander	22
They are all students of the computer science department	

Figure 147: HTML table

DITA Table Layout

The DITA table layout accepts CALS tables and simple tables.

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 148: DITA simple table

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. The following image formats are supported by default: GIF, JPEG, PNG, SVG, BMP. To extend this support for other formats (like TIFF, JPEG 2000 or WBMP, for example) *install the Java Advanced Imaging (JAI) Image I/O Tools plug-in*.

When an image cannot be rendered, Oxygen XML Author displays a warning message that contains the reason why this is happening. Possible causes:

- the image is very large and you need to enable *Show very large images* option;
- the image format is not supported by default. It is recommended to *install the Java Advanced Imaging 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.

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

Follow this procedure:

1. Start Oxygen and open the **Help** > **About** dialog. Open the **System properties** tab and look for *java.runtime.name* and *java.home* properties. Keep their values for later use.
2. Go to *Java Advanced Imaging Image I/O page* and download the kit corresponding to your operating system and Java distribution (found in the *java.runtime.name* property).
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.

Adding an Image

An image can be inserted in an XML document edited in Author mode with the following methods:

- The insert image actions from the predefined document types. The following document types have an insert image action: 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"/>
  ```

  If it is only an image part, first it is saved as a file using the file save dialog which is displayed automatically. After saving the image to a file the file path is inserted at the drop position as specified above.
- Copy the image from other application and paste it in your document. The content inserted in the document is similar with that added after dragging and dropping an image.

Editing MathML Notations

The Author editor includes a built-in editor for MathML notations. Double clicking inside a MathML notation starts the MathML editor in a new dialog where the mathematical symbols of the notation are edited.

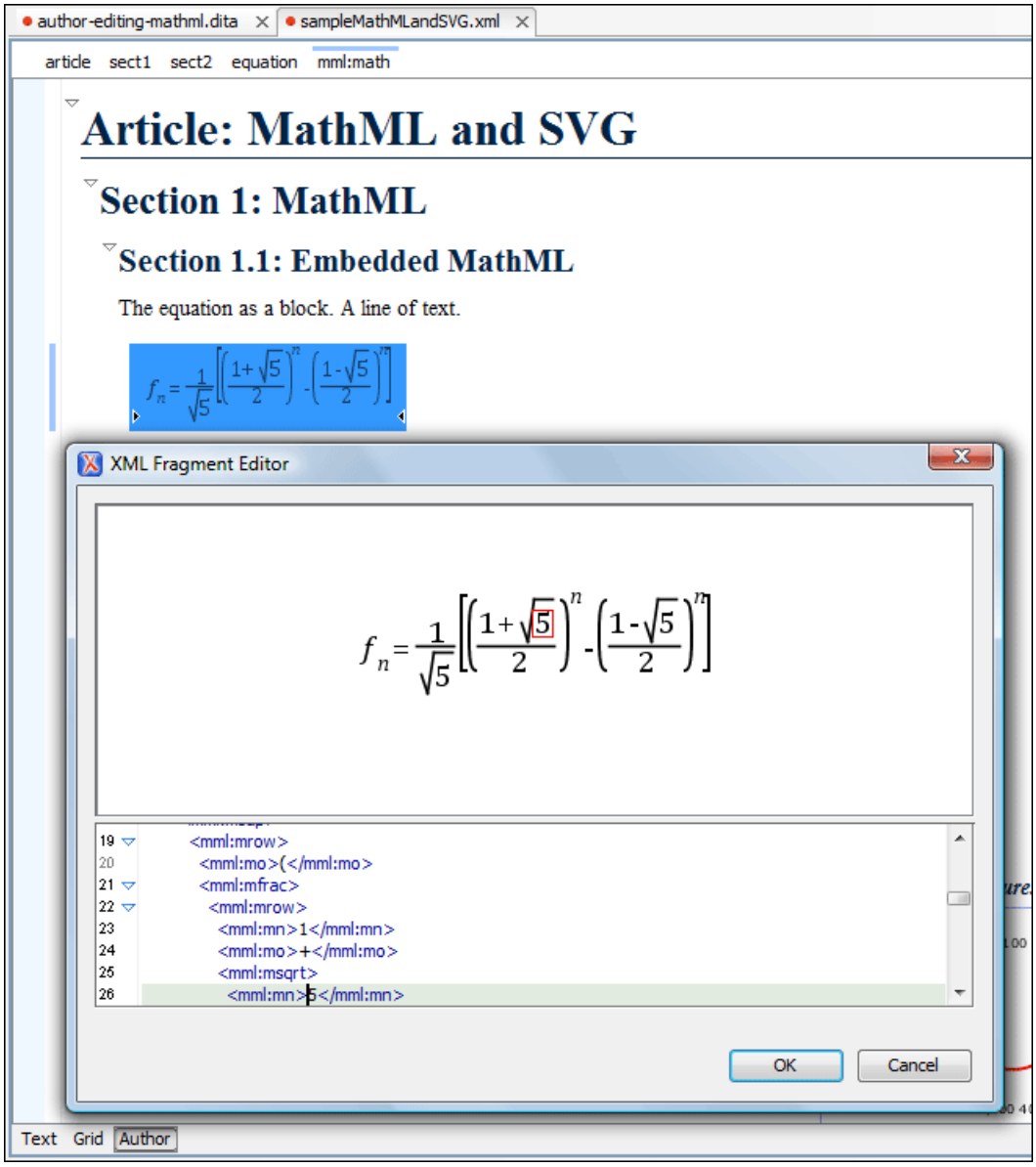


Figure 149: The default MathML editor

The minimum font size for mathematical symbols and the MathFlow SDK configuration are *set in the **Preferences***.

If a MathML file is included in the current project that is opened in the **Project** view it can be opened directly in the MathML editor with the action **Open with** > **MathFlow editor** that is available on the contextual menu of the **Project** view.

Configure the MathFlow Editor

The MathFlow Components (the MathFlow SDK) can replace the default MathML editor with a specialized MathML editor. You have to *purchase a MathML component from Design Science* and configure it in Oxygen with the following procedure:

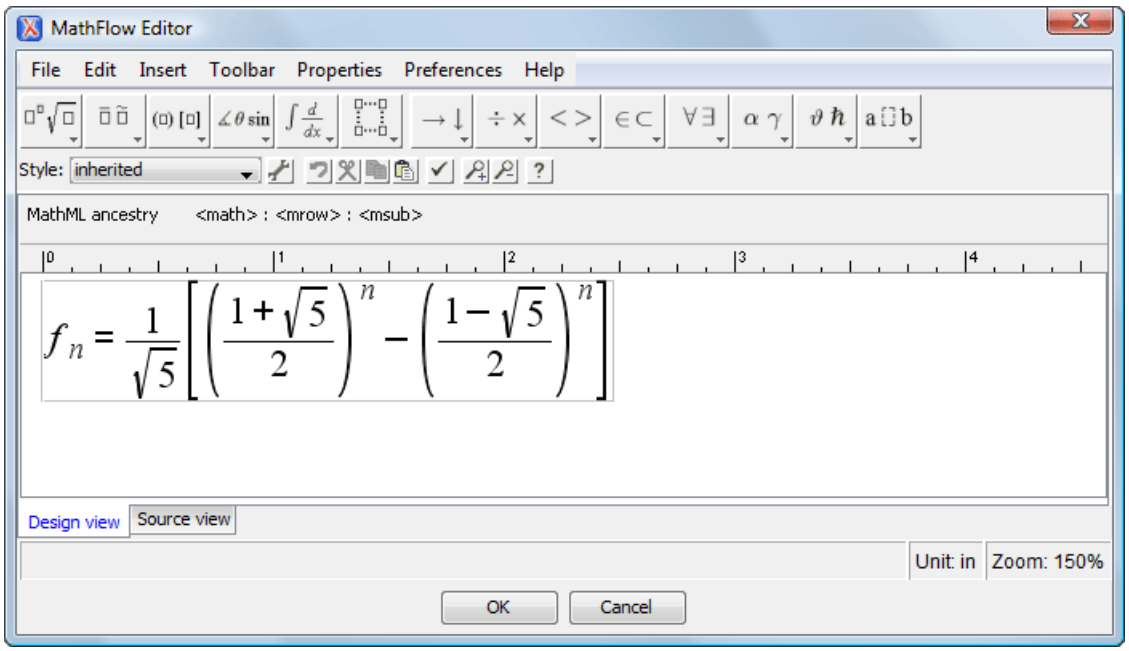


Figure 150: The default MathML editor

1. Install MathFlow Components (the MathFlow SDK).
2. On Windows make sure there is a copy of the FLEXlm DLL, that is the file `[MathFlow-install-folder]/resources/windows/lmgr10.dll`, in a folder that is added to the *PATH* environment variable.
3. Set the path to the MathFlow install folder *in the **Preferences***.
4. Set the path to the MathFlow license file *in the **Preferences***.

Refreshing the Content

On occasion you may need to reload the content of the document from the disk or reapply the CSS. This can be performed by using the **Reload** action.

For refreshing the content of the referred resources you can use the action **Refresh references**. 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*.

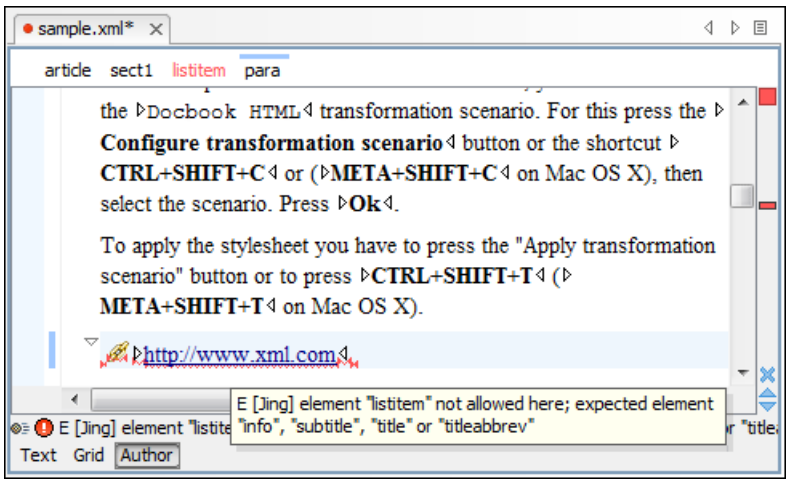


Figure 151: Error presenting in Oxygen Author editor

A fragment with a validation error or warning will be marked by underlining the error region with a red color. Also a red sign will mark the position 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 of the document is designed to display the errors found during the validation process and also to help the user to locate them more easily. The ruler contains the following areas:

- The top area - a success validation indicator that will turn green in case the validation succeeded or red otherwise.A more detailed report of the errors is displayed in the tool tip. In case there are errors, only the first three of them will be presented in the tool tip.
- The middle area - the errors markers are depicted in red (with a darker color tone for the current selected one). The number of markers shown can be limited by modifying the setting **Options** > **Preferences** > **Editor** > **Document checking** > **Limit error markers to** .

  Clicking on a marker will highlight the corresponding text area in the editor. The error message is displayed both in the tool tip and in the error area on the bottom of the editor panel.

  The *Document checking user preferences* are easily accessible from the button displayed at the beginning of the error message on the bottom of the editor panel.
- The bottom area - two navigation arrows that will go to the next or to the previous error and a button for clearing all the error markers from the ruler. The same actions can be triggered from **Document** > **Automatic validation** > **Next error (Ctrl + .)** and **Document** > **Automatic validation (Ctrl + ,)** > **Previous error (Ctrl + ,)** .

The validation status area is the line at the bottom of the editor panel that presents the message of the current validation error. Clicking on opens the ***Document checking*** page in Oxygen user preferences.

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

Whitespace Handling

There are several major aspects of white-space handling in the Oxygen Author editor 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 can insert 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 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 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.

Review

Tracking Document Changes

Track Changes is a way to keep track of the changes you make to a document. You can activate change tracking for the current document by choosing **Edit** > **Review** > **Track Changes** or by clicking the **Track Changes** button located on the Author toolbar. When *Track Changes* is enabled your modifications will be 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.

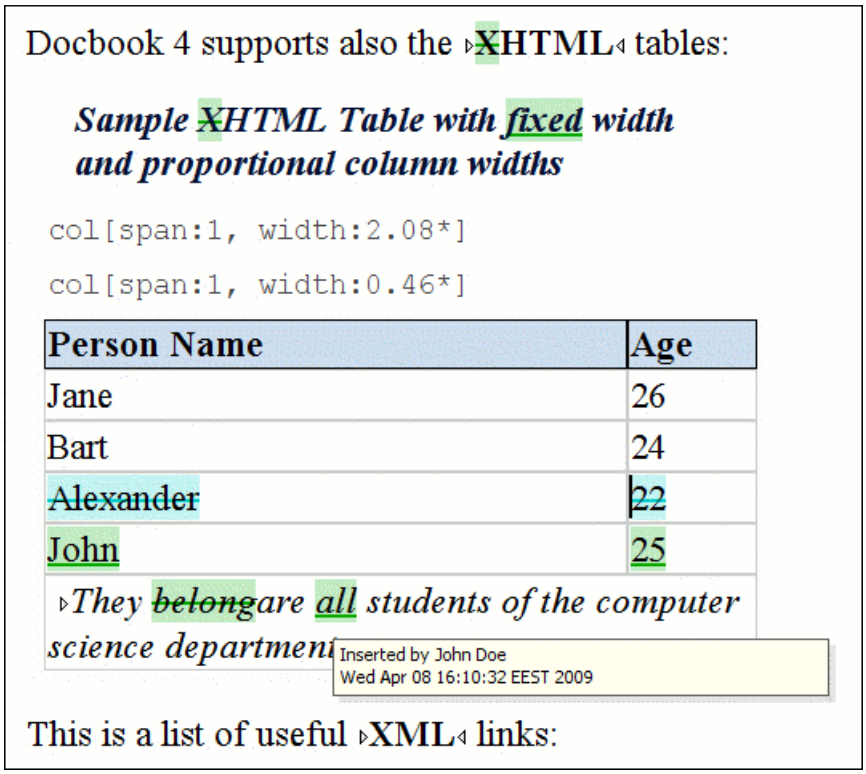


Figure 152: Change Tracking in Oxygen Author

When hovering a change the tooltip will display information about the author and modification time.

If the selection in the Author contains track changes and you copy it the clipboard will contain the selection with all the changes *accepted*. This filtering will happen only if the selection is not entirely inside a tracked change.

Tip: For each change the author name and the modification time are preserved. The changes are stored in the document as processing instructions and they do not interfere with validating and transforming it.

Adding Comments into a Document

You can associate a note or a comment to a selected area of text 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 Comments** toolbar and on the **Review** submenu of the contextual menu of Author editor.

Comments are persistent highlights with a colored background. The background color is customizable or can be assigned automatically by the application. This behaviour can be controlled from the *Review options page*.

Managing Changes

You can review the changes made by you or other authors 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 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.
- **Manage Tracked Changes** - Action designed to find and manage all changes in the current document.

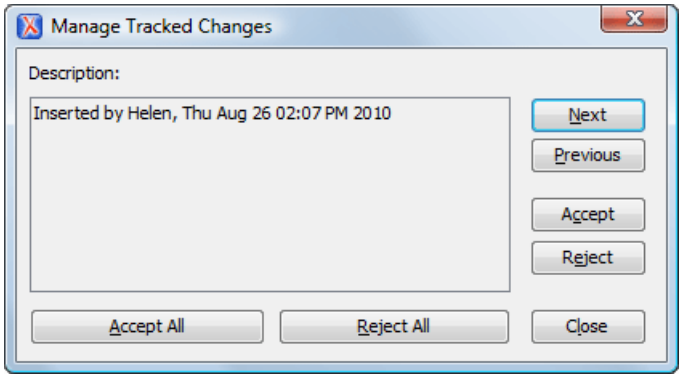


Figure 153: Manage Tracked Changes

The dialog offers the following actions:

- **Next** - Finds the next change in the document.
- **Previous** - Finds the previous change in the document.
- **Accept** - Accepts the current change. This action is also available on the contextual menu.
- **Reject** - Rejects the current change. This action is also available on the contextual menu.
- **Accept All** - Accepts all changes in the document.
- **Reject All** - Rejects all changes in the document.

The dialog is not modal and it is reconfigured after switching between the dialog and one of the opened editors.

Track Changes Visualization Modes

Three specialized actions allow you to switch between the following visualization modes:

- **View All Changes** - default visualization mode, all tracked changes are represented in the Author mode;
- **View Final** - previews the document as if all tracked changes (both inserted and deleted) were accepted;
- **View Original** - previews 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**.

👉 **Note:** All three actions are available only in a drop-down list in the **Track Changes** toolbar.

Managing Comments

A comment is marked in Author editor with a background color which can be configured for each user name.

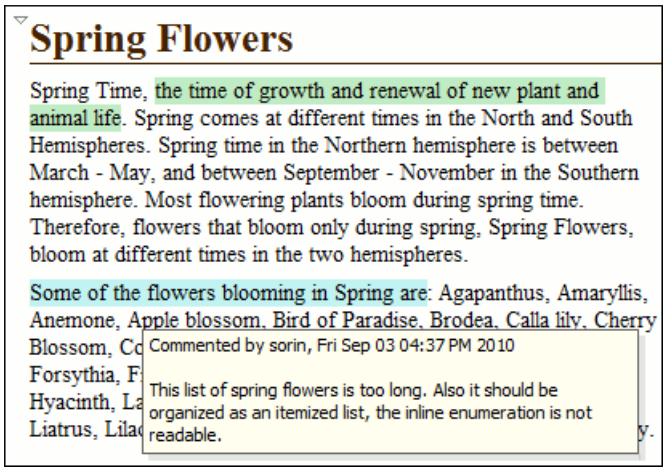


Figure 154: Manage Comments in Author Editor

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 page toolbar.
- **Edit Comment...** - Allows you to change an existing content. The action is available both on the Author page toolbar and the contextual menu.
- **Manage Comments...** - Opens a dialog that allows you to manage all comments contained in the current document. You can cycle through comments, edit, and remove individual comments or all comments. The action is available on the Author page toolbar.
- **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 page contextual menu, **Review** sub-menu.

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 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.

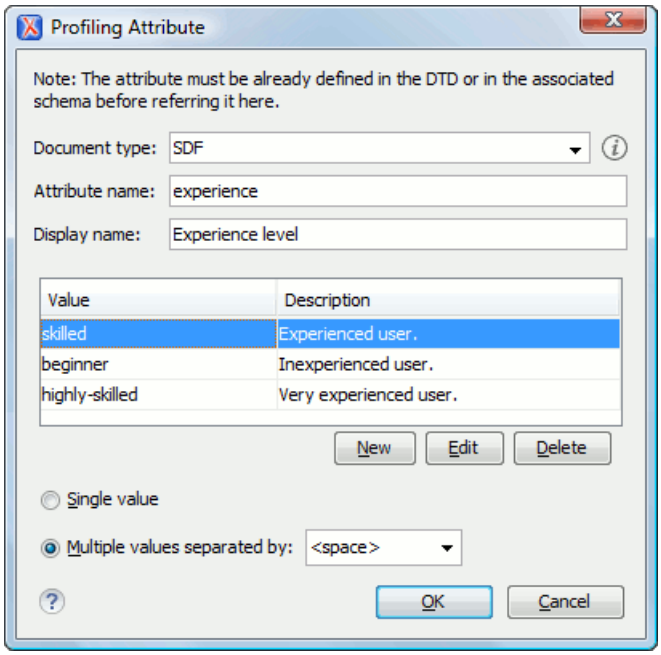
Create Profiling Attributes

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 application's toolbar **Options** > **Preferences** > **Editor** > **Pages** > **Author** menu .
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.
4. Click **OK**.
5. Click **Apply** to save the profiling attribute.

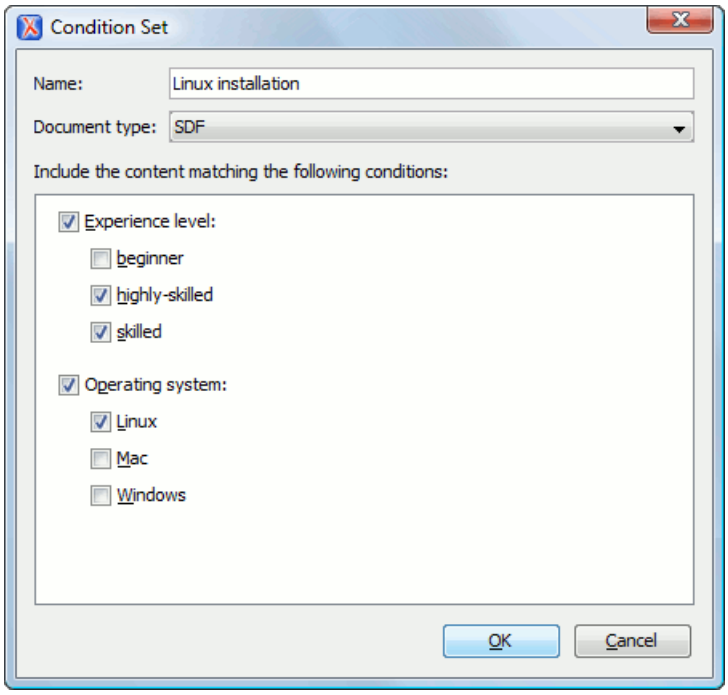
Create Profiling Condition Sets

Several profiling attributes can be aggregated into a profiling condition set that allow 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 application's toolbar **Options** > **Preferences** > **Editor** > **Pages** > **Author** menu .
2. In the **Profiling Condition Sets** area, press the **New** button.

   The following dialog is opened:



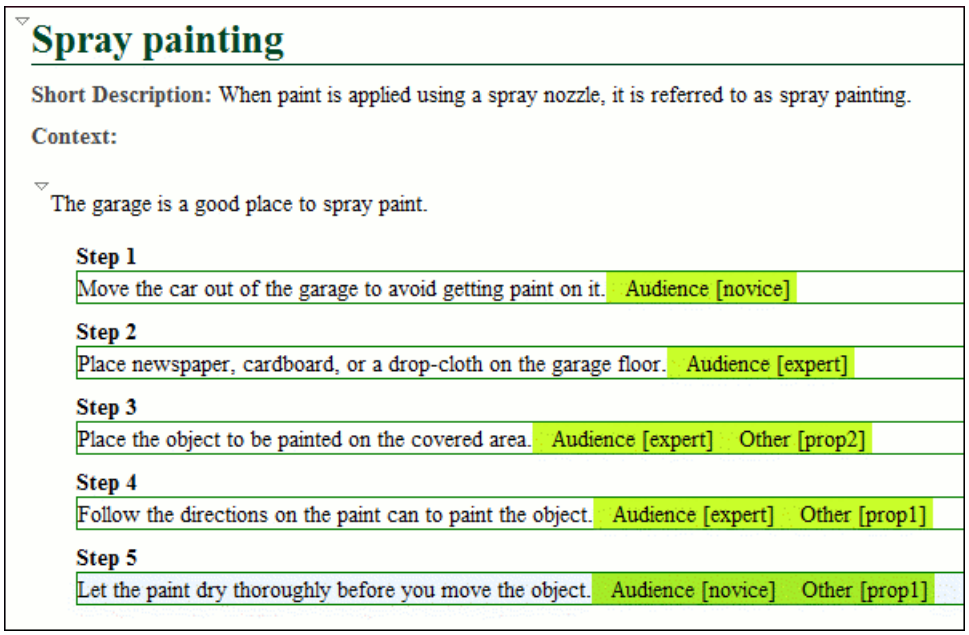
3. Fill-in the dialog as follows:
   a) Type the condition set's name.
   b) Choose the document type for which you have previously defined profiling attributes.
   After choosing a document type, all profiling attributes and their possible values are listed in the central area of the dialog.
   c) Define 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

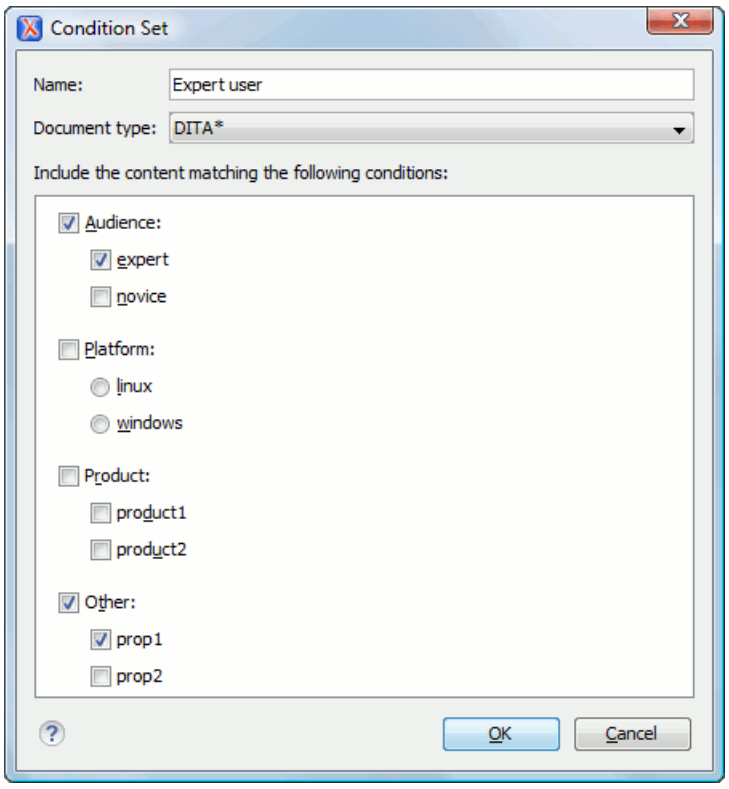
All defined Profiling Condition Sets are available as shortcuts in the Profiling / Conditional Text menu. Just click on a menu entry to apply the condition set. The filtered content is grayed-out.

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.

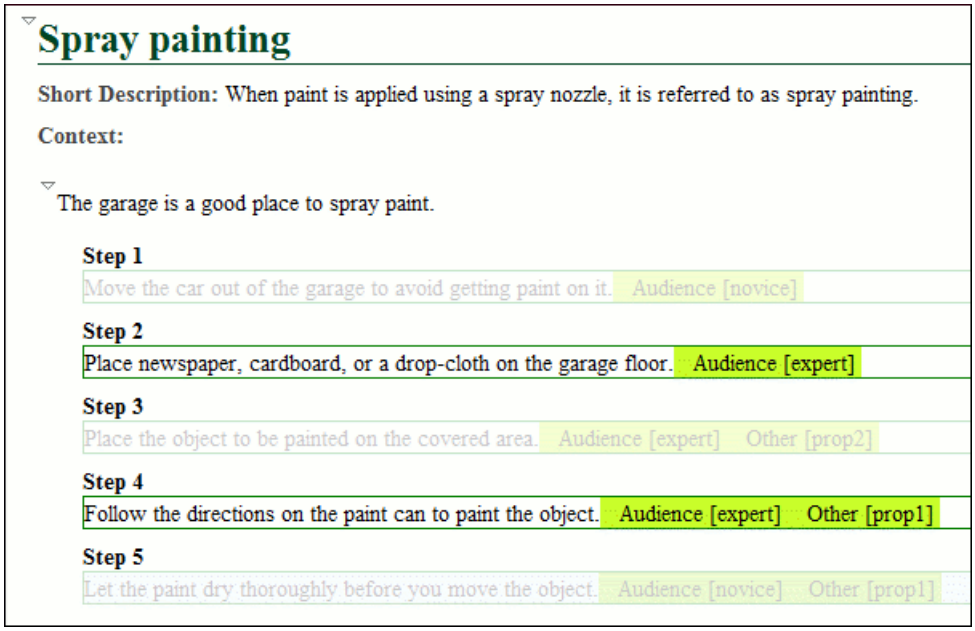
As an example, let us suppose that you have the following document:



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:



Apply Profiling Attributes

Profiling attributes are applied on element nodes.

To set a profiling attribute:

1. Invoke the application's contextual menu. Click the **Edit Profiling Attributes...** action.
   The displayed dialog shows all profiling attributes and their values, defined on the document type of the edited content.
2. In the **Edit Profiling Attributes** dialog, tick the checkboxes corresponding to attribute values you want to apply on the current element.
3. 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** page. Also, all profiling attributes set on the current element are listed at the end of the highlighted block and in its tooltip message.

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.

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*
- *Reusing Content*
- *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 provides a special view for editing DITA maps. The **DITA Maps Manager** view presents a map in a simplified table-of-contents manner allowing the user to navigate easily to the referred topics and maps, make changes and perform transformations to various output formats using the DITA-OT framework bundled with Oxygen.

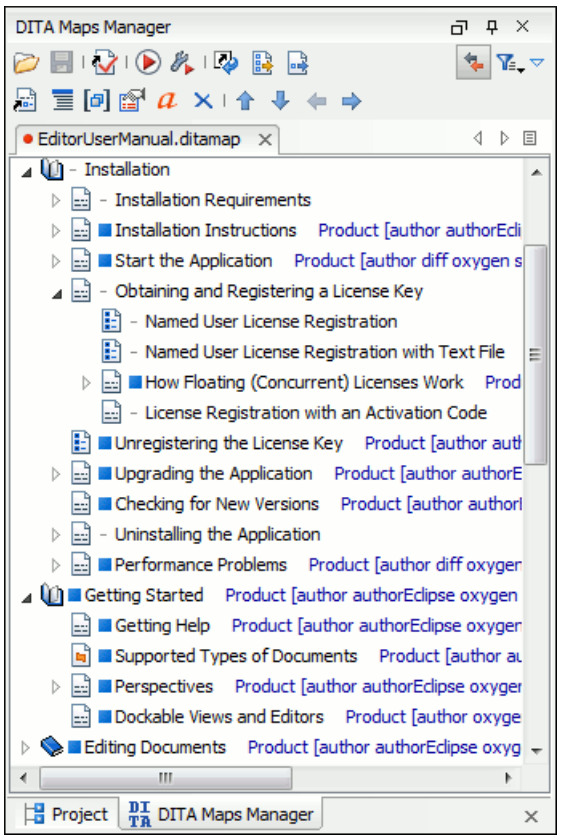


Figure 155: The DITA Maps Manager View

The view supports opening multiple documents at the same time.

All files which have the extension `.ditamap` or `.bookmap` are open in the **DITA Maps Manager** view. In addition, you can right click the file in the **Project** view and select **Open with**. If you opened the DITA map file in the editor panel you can open it also in this special DITA map view by right clicking on the title tab of the editor panel and selecting the **Open in DITA Maps Manager** action from the popup menu.

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.
- **Save (Ctrl+S)** - Saves the current DITA map.
- **Validate and Check for Completeness** - *Checks the validity and integrity* of the map.
- **Apply Transformation Scenario** - *Applies the DITA map transformation scenario* that is associated with the current map from the view.
- **Configure Transformation Scenario** - 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 Author, the DITA Map is updated automatically.
- **Open Map in Editor with Resolved Topics** - Opens the result of expanding all topic references in Author editor.
- **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.
- **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.
- **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 panel for adding and editing the profiling conditions that can be applied in the **DITA Maps Manager** view and the **Author** editor.

Tip: 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 246 topic.
- **Edit Properties** - Edit the properties of a selected node. You can find more details about this action in the *Edit Properties* on page 246 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 213 topic.
- **Delete** - Deletes the selected node.
- **Move Up (Alt + Up)** - Moves the selected nodes in front of their respective previous siblings.
- **Move Down (Alt + Down)** - Moves the selected nodes after their next respective siblings.
- **Promote (Alt + Left)** - Moves the selected nodes after their respective parents as a siblings.
- **Demote (Alt + Right)** - Moves the selected nodes as children to their respective previous siblings.

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 tries to resolve both cases. `keyref`s are solved relative to the current map.

In additional to being available on the toolbar and on the contextual menu, more navigation actions and all edit actions appear in the **DITA Maps** menu. The menu is only available when the view is active on screen.

The contextual menu contains, in addition to the edit actions described above, the following actions:

- **Open in Editor** - Opens in the editor the resources referred by the selected nodes.
- **Append Child/Insert After** - Sub-menus containing the following actions:
  - **Reference** - Appends/Inserts a topic reference as a child/sibling of the selected node.
  - **Reference to the current edited file** - Appends/Inserts a topic reference to the current edited file 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.
  - **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.
- **Find/Replace in Files** - Find and replace in files using the scope of the current edited map.
- **Check Spelling in Files** - Checks the spelling of the files in the scope of the current edited map.
- **Open Map in Editor with resolved topics** - Opens the map in the main editing area with all the topic references expanded in the map content.
- **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 **PromoteAlt+Left Arrow** and **DemoteAlt+Right Arrow** operations.

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 page 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

The validation of DITA maps is done with the action **Validate and Check for Completeness** that is available on *the **DITA Maps Manager** view* toolbar and on the **DITA Maps** menu.

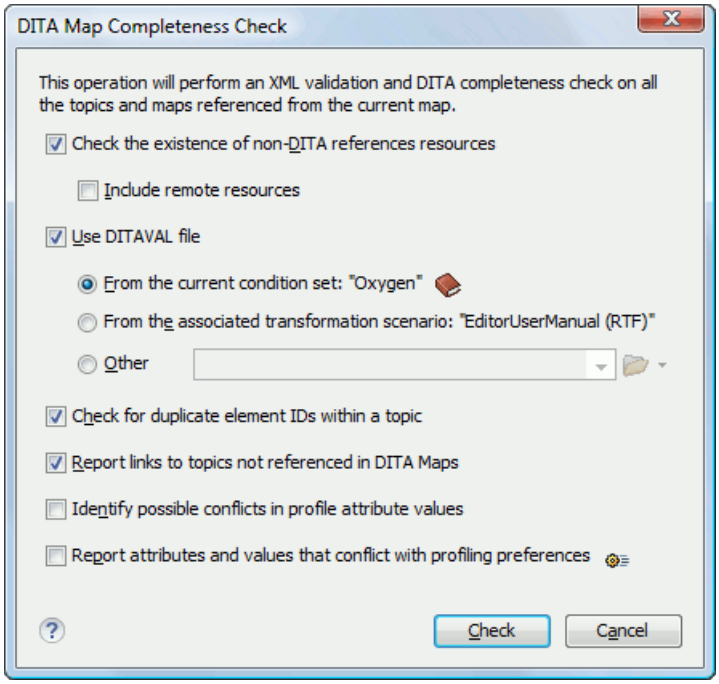


Figure 156: DITA Map Completeness Check

The validation process does the following:

- Checks the file paths of the topic references. If a `href` attribute points to an invalid file path it is reported as a separate error in the **Errors** view.
- Validate each referred topic and map. Each topic file is opened and validated against the appropriate DITA DTD. If other map is referred in the main map, it is checked recursively applying the same algorithm as for the main map.

You can customize the operation setting the following options:

- **Check the existence of non-DITA references resources** - extends the validation of referred resources to non-DITA files. You can also choose to include in the validation also the remote resources;

- **Use DITAVAL file** - profiling conditions are applied. First the content of the map is filtered by applying a *profiling condition set*. The condition set can be either the one applied currently in the DITA Maps Manager view (the radio button **From the current condition set**) or the one specified explicitly as a DITAVAL file in the current transformation scenario associated with the DITA map (the radio button **From the associated transformation scenario**). If a link is invalid in the content that resulted from the filtering process then it is reported as 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.

Create a Topic in a Map

You add a new topic to a DITA map with the following 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 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 `bookmap` 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.

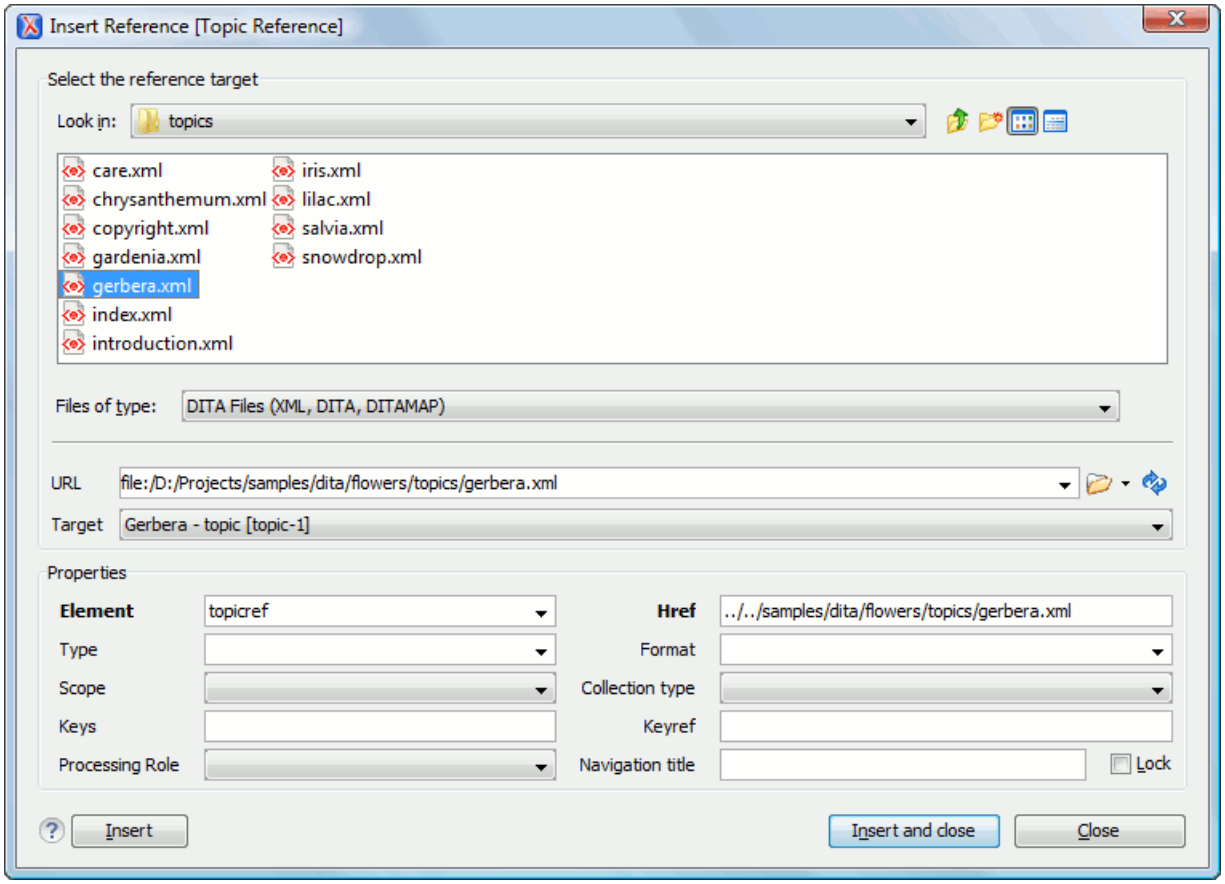


Figure 157: Insert Topic Reference Dialog

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.

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.

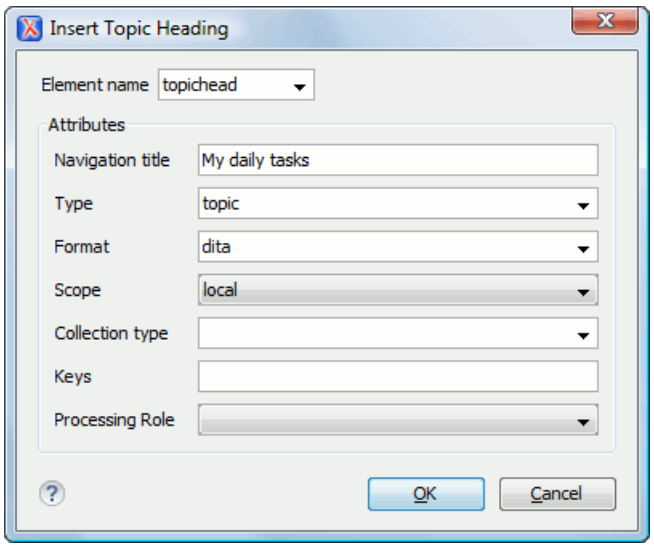


Figure 158: Insert Topic Heading Dialog

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.

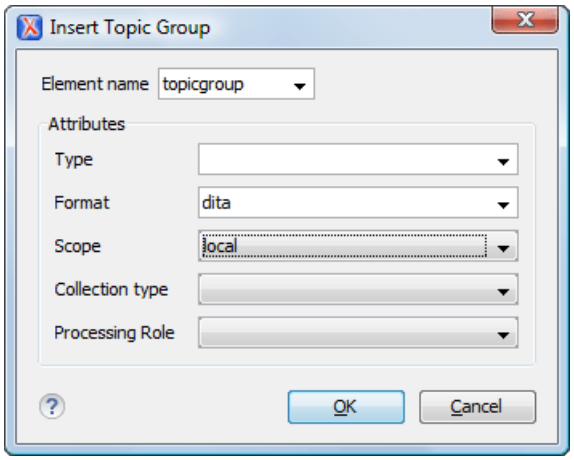


Figure 159: Insert Topic Group Dialog

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 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 1.5.2 and ANT 1.7 come bundled in Oxygen.

More information about the DITA Open Toolkit are available at *http://dita-ot.sourceforge.net/*.

Available Output Formats

You can publish DITA-based documents in any of the following formats:

- **PDF** - DITA to PDF using the DITA OT IDIOM PDF plugin.
- **WebHelp** - DITA to XHTML.
- **XHTML** - DITA to XHTML.
- **Compiled HTML Help (CHM)** - DITA Map to HTML Help. If HTML Help Workshop is installed on your computer, then Oxygen detects it and uses it to perform the transformation. When the transformation fails, the hhp (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 *.hhp* and *.hhc* files.
- **JavaHelp** - DITA Map to JavaHelp.
- **Eclipse Help** - DITA Map to Eclipse Help.
- **Eclipse Content** - DITA Map to Eclipse Content.
- **TocJS** - A JavaScript file that can be included in an HTML file to display in a tree-like manner the table of contents of the transformed DITA map.
- **Open Document Format** - DITA Map to ODF.
- **Docbook** - DITA Map to Docbook.
- **RTF** - DITA Map to Rich Text Format.
- **troff** - DITA Map to Text Processor for Typesetters.
- **Legacy PDF** - DITA Map to PDF using the DITA OT deprecated PDF implementation.

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 `${frameworks}/dita/DITA-OT/demo/tocjs/basefiles` folder in the transformation's output folder.
3. Copy the `${frameworks}/dita/DITA-OT/demo/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"`.

WebHelp Output Format

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include continuous content update and platform independence, since it can be viewed using a regular web browser.

Oxygen allows you to publish a DITA Map into a WebHelp format that provides both Table of Contents and advanced search capabilities.

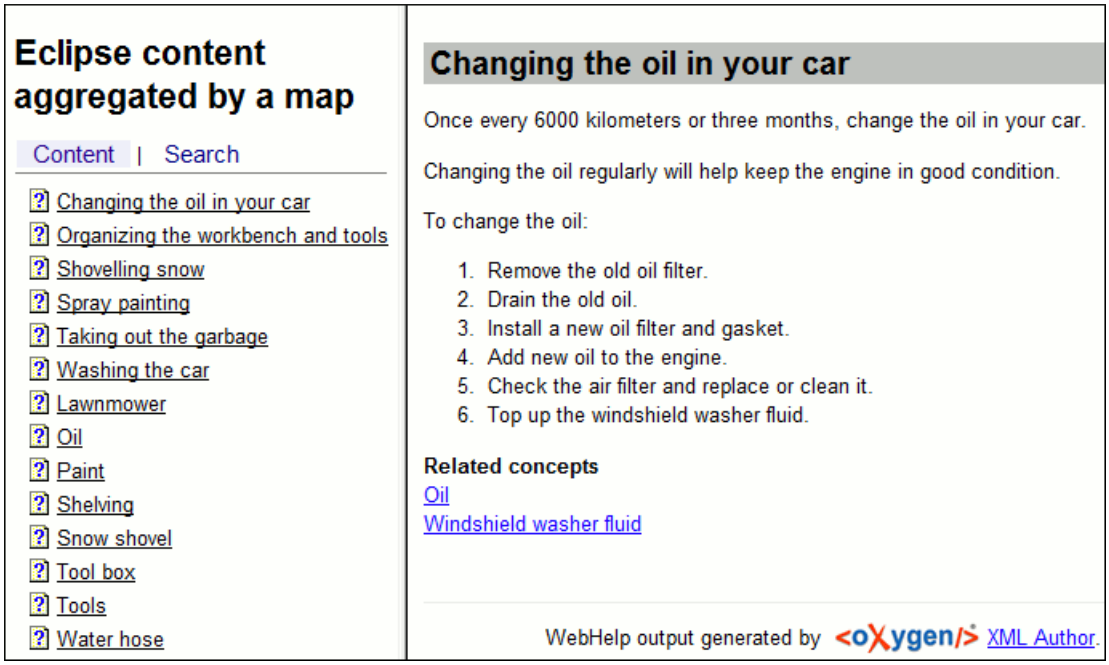


Figure 160: WebHelp Output

The layout is composed of two frames:

- left frame, containing separate tabs for Table of Contents and Search;
- central frame where help pages are displayed.

To publish the DITA map to WebHelp, you can use the **DITA Map WebHelp** transformation. You can further 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 or not 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. Default setting is `no`.

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 emphasised section of text it scores better than a word found in a unformatted text.

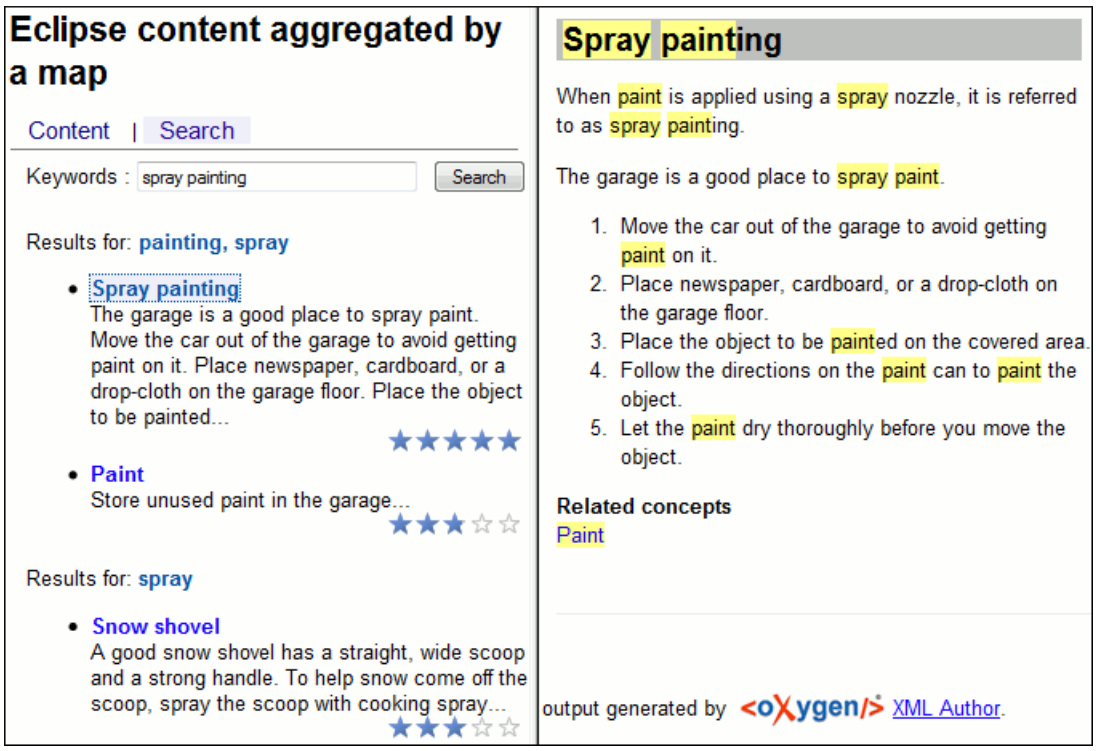


Figure 161: WebHelp Search with Stemming Enabled

Rules applied during search:

- keywords are separated by the space character. An expression like *spray painting* counts as two separate keywords: *spray* and *painting*.
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"spray painting"*, returns no results in our case, because it searches for two separate words: *"spray* and *painting"* (note the quote signs attached to each word).

Configuring a DITA Transformation

Creating map transformation scenarios is similar to creating scenarios in the main editing area.

The ***Configure Transformation Scenario** dialog* is opened from the toolbar action **Configure Transformation Scenario** of the ***DITA Maps Manager view***. Select as scenario type **DITA OT transformation** then press the **New** button. Next step involves choosing the type of output the DITA-OT ANT scenario will generate:

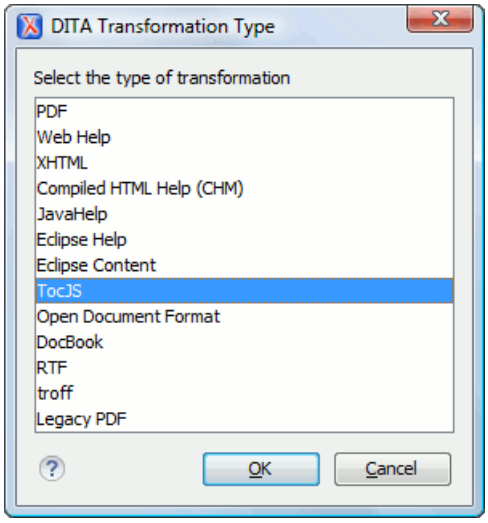


Figure 162: Select DITA Transformation type

Depending on the chosen type of output Oxygen generates values for the default ANT parameters so that you can execute the scenario right away without further customization.

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 from Oxygen are also used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the *HTTP/Proxy Configuration*.

Customizing a DITA Scenario

This section explains how to edit the parameters of a DITA transformation in the dialog box for configuring such a transformation.

The *Parameters* Tab

In the scenario **Parameters** tab you can customize all the parameters which will be sent to the DITA-OT build file.

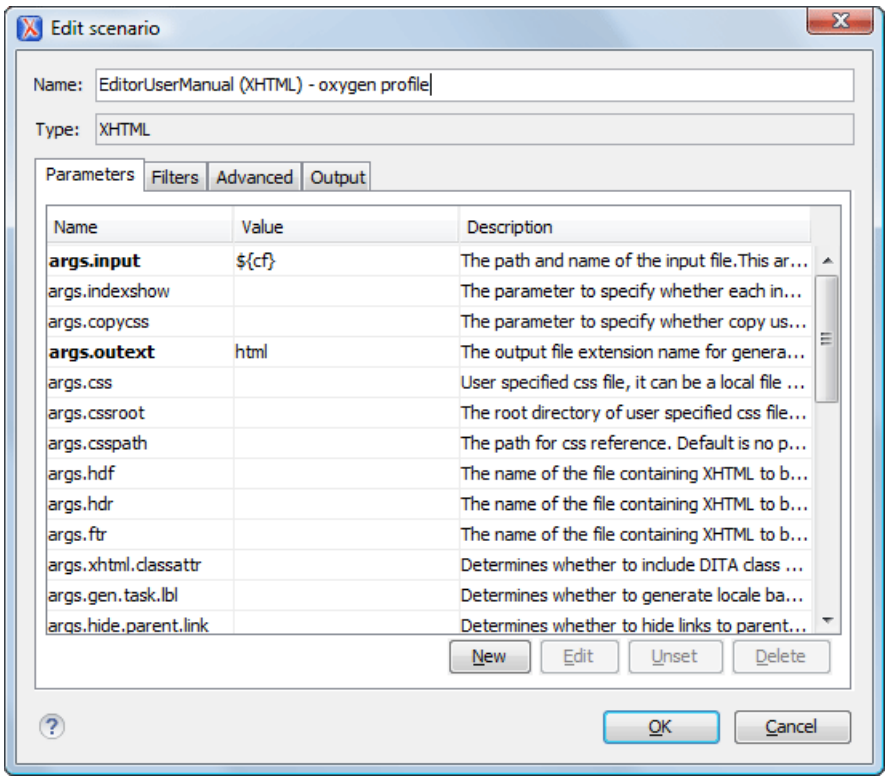


Figure 163: Edit DITA Ant transformation parameters

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 parameter type the parameter value will be:

- 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:

- `${ask('message')}` - Only the message displayed for the user is specified.
- `${ask('message', generic, 'default')}` - `'message'` will be displayed for the user, the type is not specified (the default is string), the default value will be `'default'`.
- `${ask('message', password)}` - `'message'` will be displayed for the user, the characters typed will be replaced with a circle character.
- `${ask('message', password, 'default')}` - Same as above, default value will be `'default'`.
- `${ask('message', url)}` - `'message'` will be displayed for the user, the type of parameter will be URL.
- `${ask('message', url, 'default')}` - Same as above, default value will be `'default'`.

The *Filters* Tab

In the scenario **Filters** tab you can add filters to remove certain content elements from the generated output.

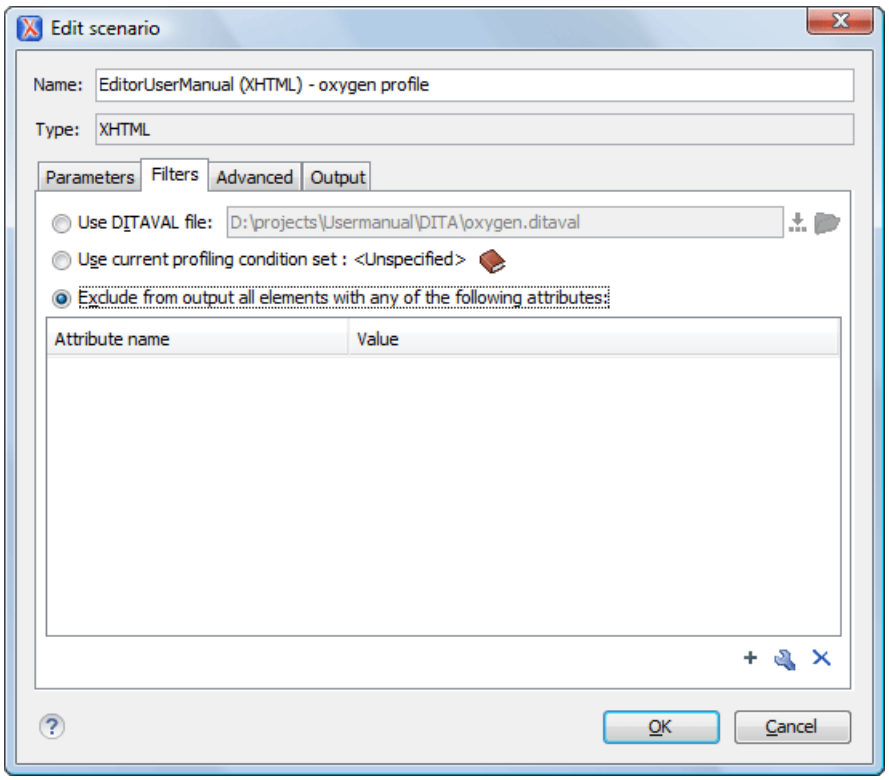


Figure 164: Edit Filters tab

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 current profiling condition set** - Allows you to use the currently active profiling condition set.
- **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.

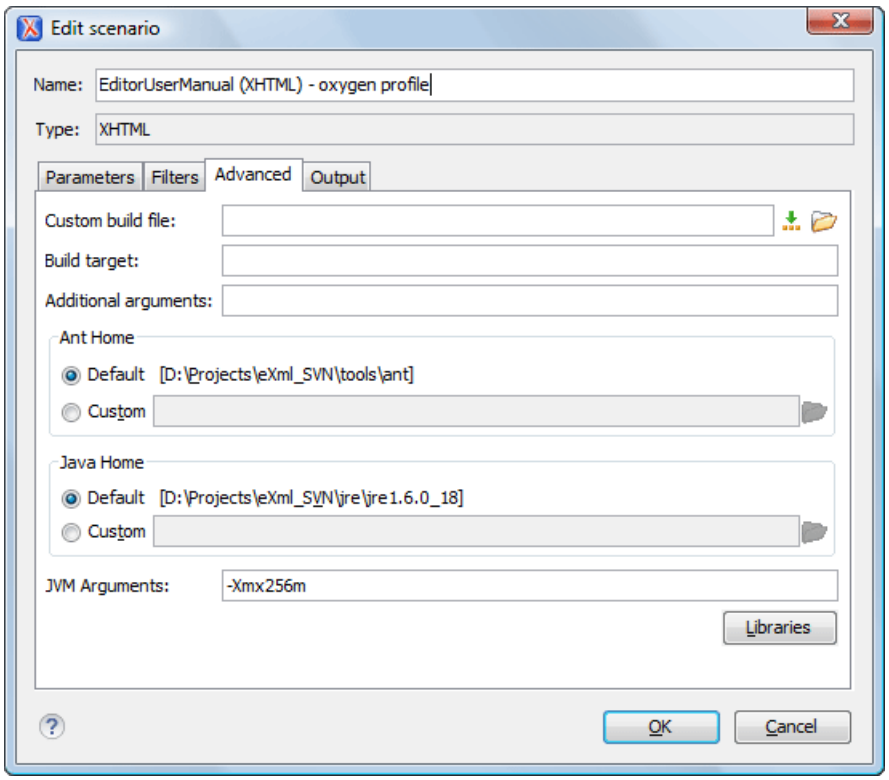


Figure 165: Advanced settings tab

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.
- **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.
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to -Xmx256m which means the transformation process is allowed to use 256 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 (256 MB) to a higher value, like 512 MB. In 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, open the *Parameters tab* and increase the value of `maxJavaMemory` parameter (default value is *500*).

- **Libraries** - Oxygen 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 will be generated.

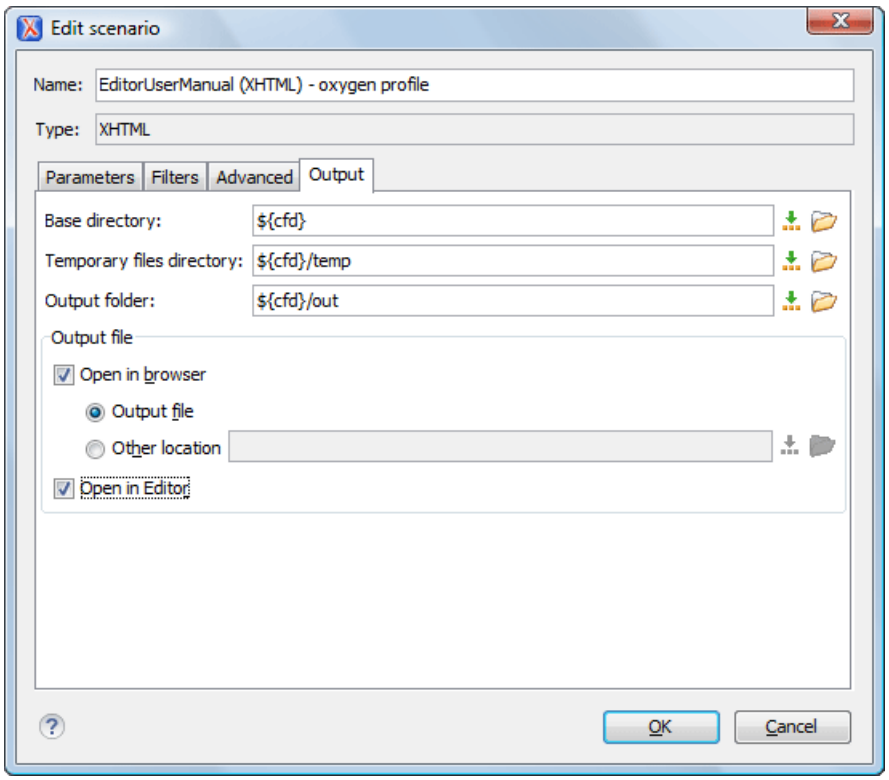


Figure 166: Output settings tab

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 will be used to store pre-processed temporary files until the final output is obtained.
- **Output folder** - The folder where the final output content will be copied.
- **Output file options** - The transformation output can then be opened in a browser or even in the editor if specified.

The *FO Processor* Tab

This tab appears only when selecting to generate PDF output using the IDIOM FO Plugin and allows you to choose the FO Processor.

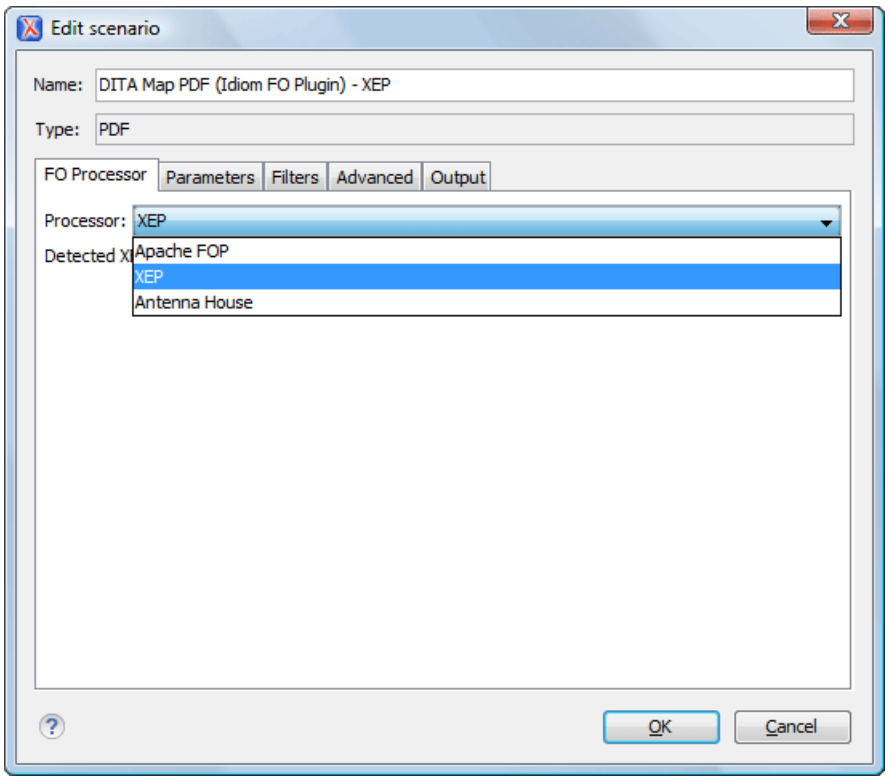


Figure 167: FO Processor configuration tab

You can choose between three processors:

- **Apache FOP** - This processor comes bundled with Oxygen.
- **XEP** - The *RenderX* XEP processor.

  If you select **XEP** in the combo and XEP was already installed in Oxygen you can see the detected installation path appear under the combo.

  XEP is considered as installed if it was detected from one of the following sources:

  - XEP was added as an external FO Processor in the Oxygen preferences.
  - 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/demo/fo/lib` directory of the Oxygen installation directory.

- **Antenna House** - The *Antenna House* AH (v5) or XSL (v4) Formatter processor.

  If you select **Antenna House** in the combo and Antenna House was already installed in Oxygen you can see the detected installation path appear under the combo.

  Antenna House is considered as installed if it was detected from 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 preferences.

Tip: The DITA-OT contributors recommend the use of the IDIOM FO Plugin to transform DITA Maps to PDF as opposed to using the standard PDF target in the DITA-OT framework. As IDIOM is also bundled with Oxygen the *PDF2 - IDIOM FO Plugin* output format should be your first choice in transforming your map to PDF. If you do not have a commercial license for XEP or Antenna House you can transform using the Apache FO Processor.

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*.

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.

Customizing the Oxygen Ant Tool

The Ant 1.7 tool which comes with Oxygen is located in the `[Oxygen-install-folder]/tools/ant` directory. Any additional libraries for Ant must be copied to the Oxygen Ant `lib` directory.

If you are using Java 1.6 to run Oxygen the Ant tool should need no additional libraries to process JavaScript in build files.

If you are using Java 1.5 you have to copy the *bsf.jar* and *js.jar* libraries in the `[Oxygen-install-folder]/tools/ant` directory.

Upgrading to a New Version of DITA OT

The DITA OT framework bundled in Oxygen is located in the `[Oxygen-install-folder]/frameworks/dita/DITA-OT` folder.

Important: There are a couple of modifications made to the DITA OT framework which will be overwritten if you choose to copy the new DITA-OT version over the bundled one:

- The DTD's in the framework have been enriched with documentation for each element. If you overwrite you will lose the documentation which is usually shown when hovering an element or in the **Model** view.
- Several build files from the IDIOM plugin have been modified to allow transformation using the Oxygen Apache built-in FOP libraries and usage of the Oxygen classpath while transforming.
- Oxygen provides Java patches for some DITA OT problems. These patches are located in the `[Oxygen-install-folder]/frameworks/dita/DITA-OT/lib/dost-patches.jar` library. If the patches library conflicts with the new DITA OT libraries it either has to be removed from disk or removed from the libraries list available in the DITA Map transformation scenario.

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*. 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 Specialization Support

This section explains how you can integrate and edit a DITA specialization in Oxygen XML Author.

Integration of a DITA Specialization

A DITA specialization 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 Oxygen XML Author with minimum effort.

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

Oxygen 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. If you want to use exclusively the libraries that come with the new DITA Open Toolkit you have to go to the **Advanced** tab, click the **Libraries** button, uncheck the checkbox **Allow Oxygen to add high priority libraries to classpath** and configure all libraries that will be used by the ANT process.
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 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`.

If the transformation fails after completing these steps please also take a look at this note about *upgrading the DITA OT which comes bundled with Oxygen*.

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.
- 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** > **Pages** > **Author** .

Working with Content References

The DITA feature called `conref` (short for *content reference*) enables a piece of content to be included by reference in multiple 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.

  This strategy requires more setup than reusable components, but makes easier centrally managing the reused content.

Oxygen XML Author 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 Author 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.

How to Work with Reusable Components

When you need to reuse a part of a DITA topic in different places (in the same topic or in different topics) it is recommended to create a separate component and insert only a reference to the new component in all places. Below are the steps for extracting a reusable component, inserting a reference to the component and quickly editing the content inside the component.

1. Select with the mouse the content that you want to reuse in the DITA file opened in Author mode.
2. Start the action **Create Reusable Component** that is available on the DITA menu, the Author framework actions toolbar and the contextual menu of the Author editor.
3. In the combo box **Reuse Content** select the DITA element with the content that you want to extract in a separate component. The combo box contains the current DITA element where the cursor is located (for example a `p` element - a paragraph - or a `step` or a `taskbody` or a `conbody` etc.) and also all the ancestor elements of the current element.
4. In the **Description** area enter a textual description for quick identification by other users of the component.
5. If you want to replace the extracted content with a reference to the new component you should leave the checkbox **Replace selection with content reference** with the default value (selected).
6. Press the **Save** button which will open a file system dialog 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 should follow these steps for inserting an element with a content reference (`conref`) attribute:

1. Start one of the actions **Insert a DITA Content Reference** and **Insert a DITA Content Key Reference**.
2. In the dialog **Insert Content Reference** select the file with the referenced content in the **URL** field.
3. In the tree that presents the DITA elements of the specified file that have an `id` attribute you have to select the element or the interval of elements that you want to reference. The `conref` field will be filled automatically with the `id` value of the selected element. If you select an interval of elements the `conrefend` field will be filled with the `id` value of the element that ends the selected interval.
4. Press the **OK** button to insert in the current DITA file an element with the same name and with the same `conref` attribute value (and optionally with the same `conrefend` attribute value) as the element(s) selected in the dialog.

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 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 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 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.

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** page with a light green border.

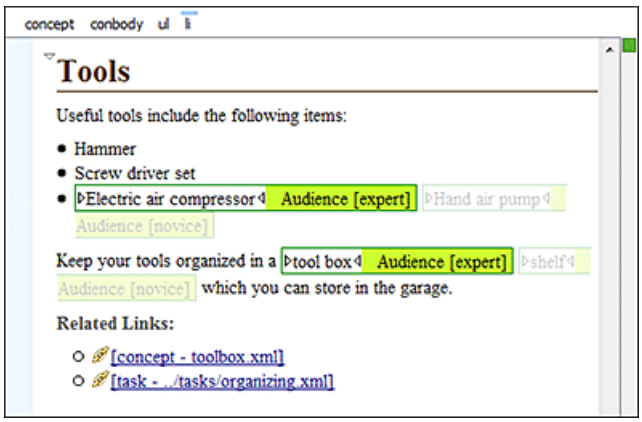


Figure 168: Profiling in Author

In the ***DITA Maps Manager View*** view different decorators are user 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.

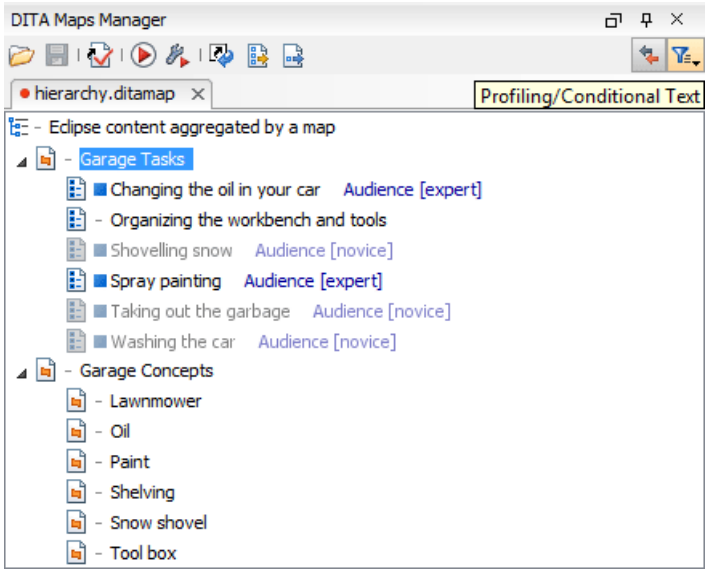
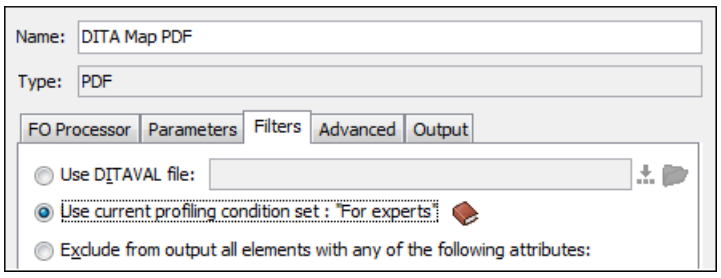


Figure 169: Profiling in DITA Maps Manager

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.

Publish Profiled Text

Oxygen comes with preconfigured transformation scenarios for DITA. All these scenarios take into account the current profiling condition set.



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.

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 P4 Document Type*
- *The TEI P5 Document Type*
- *The MathML Document Type*
- *The Microsoft Office OOXML Document Type*
- *The Open Office ODF Document Type*
- *The OASIS XML Catalog Document Type*
- *The XML Schema Document Type*
- *The Relax NG Document Type*
- *The NVDL Document Type*
- *The Schematron Document Type*
- *The Schematron 1.5 Document Type*
- *The XSLT Document Type*
- *The XMLSpec Document Type*
- *The FO Document Type*
- *The EAD Document Type*
- *The EPUB Document Type*

This chapter includes short presentations of the document types that come bundled with Oxygen. For each document type there are enumerated the 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 Tagless editor 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 the application.

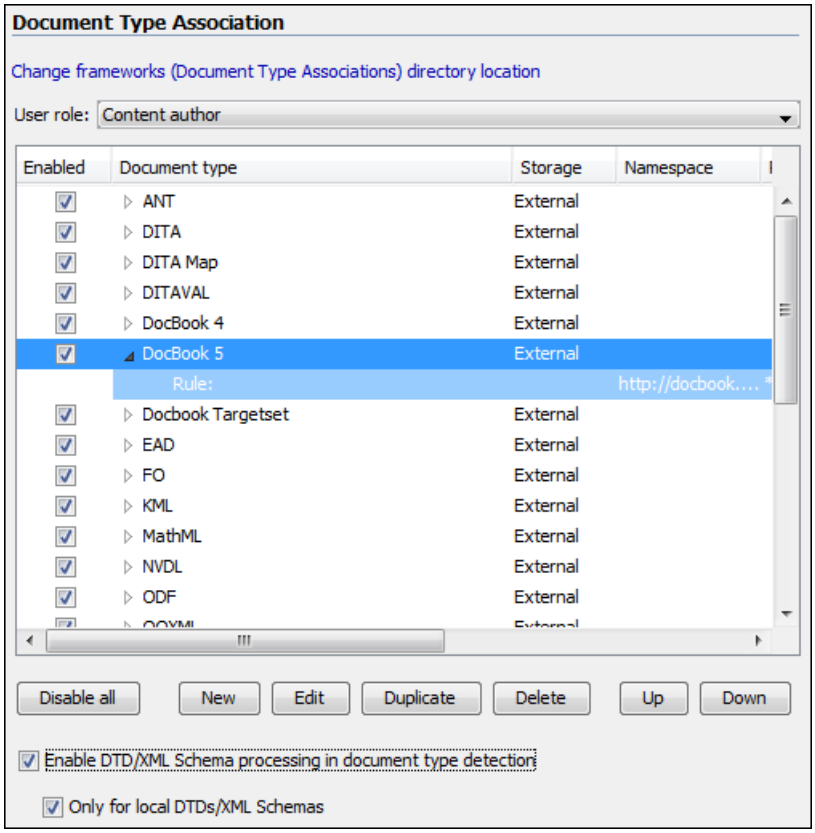


Figure 170: Document Type preferences page

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 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`.

Author Extensions

Specific actions for DocBook documents are:

- **Bold emphasized text** - Emphasizes the selected text by surrounding it with `<emphasis role="bold"/>` tag.
- **Italic emphasized text** - Emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag.
- **Underline emphasized text** - Emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag.

  Note: 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.

- **link** - Inserts a hypertext link.
- **ulink** - Inserts a link that addresses its target with an URL (Universal Resource Locator).
- **olink** - Inserts a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values which are present in a ***Targetset*** file.

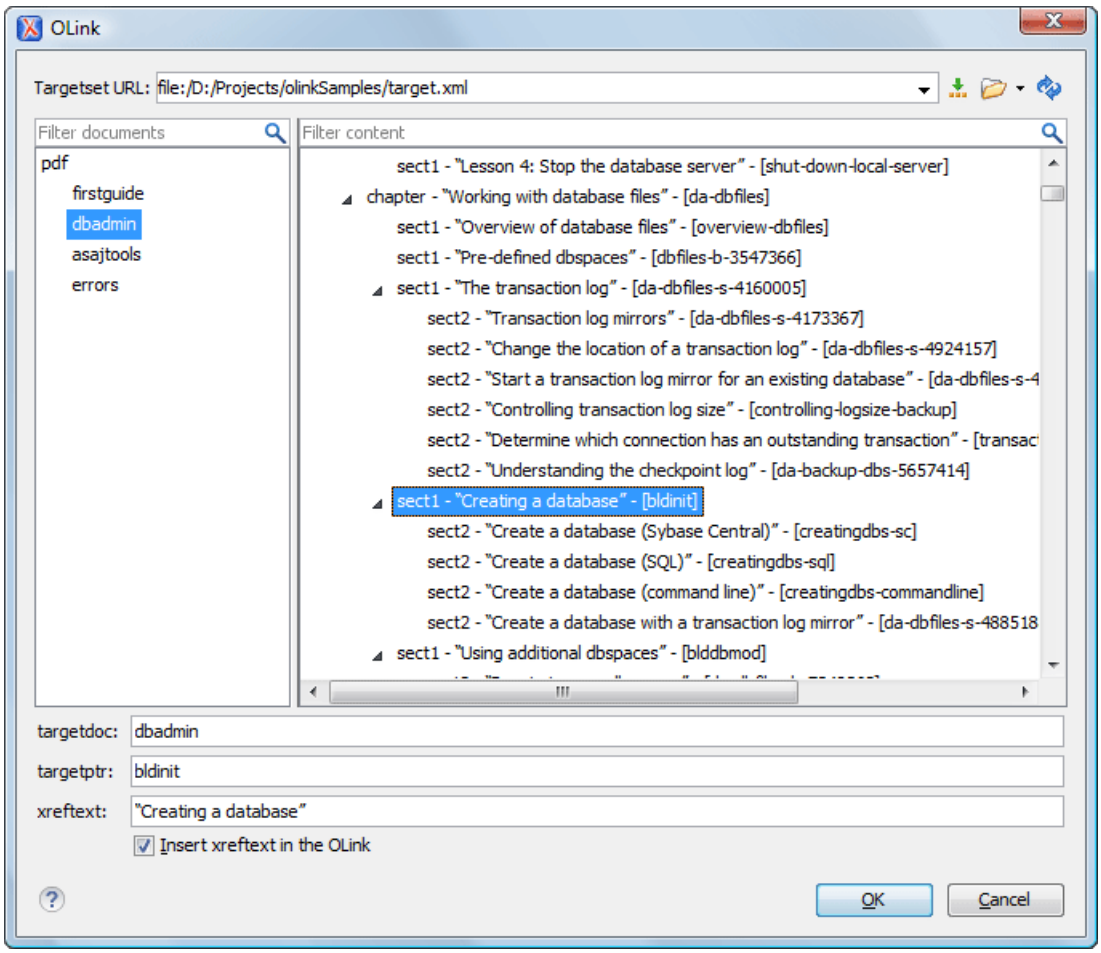


Figure 171: Insert OLink Dialog

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.

- **URI** - Inserts an URI element. The URI identifies a Uniform Resource Identifier (URI) in content.
- **xref** - Inserts a cross reference to another part of the document.

 **Note:** These actions are grouped under the **Link** toolbar actions group.

- § **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.
- ¶ **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `para`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
-  **Insert Graphic** - 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 Ordered List** - Inserts an ordered list. A child list item is also inserted automatically by default.
-  **Insert Itemized List** - Inserts an itemized list. A child list item is also inserted automatically by default.

- **Insert Variable List** - Inserts a DocBook variable list. A child list item is also inserted automatically by default.
- **Insert List Item** - Inserts a new list item in any of the above three list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table, if the header and footer will be generated and how the table will be framed. **CALS** or **HTML** table model can be selected.

  Note: If the **Title** checkbox is unchecked, an `informaltable` element is inserted.

- **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.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one is inserted at caret position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **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 cell from current caret position with that of the cell above it. This action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. This action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column. The column span of the source cell is decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column. The column span of the source cell is decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above it. This action works only if the current cell spans over more than one row. The row span of the source cell is decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below it. This action works only if the current cell spans over more than one row. The row span of the source cell is decreased with one.

Note: DocBook 4 supports only CALS table model. HTML table model is supported only in DocBook 5.

Caution: Column specifications are required for table actions to work properly.

- **Generate IDs** - Allows generating ID for the current selection or for the element at caret position if the element appears in **ID Generation** dialog.

  In this dialog, you can specify the elements for which Oxygen generates an ID. You can choose to automatically generate an ID for these elements by selecting **Auto generate ID's for elements**. You can choose a pattern for the generated ID using the field **ID Pattern**. If the element already has an ID, this ID is preserved.

All actions described above are available in the contextual menu, the **DocBook4** submenu of the main menu and 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).

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 (experimental) and EPUB.

Templates

Default templates are available in the *New File wizard* and can be used for easily creating a skeletal form of a DocBook 4 book or article. These templates are stored in the `${frameworks}/docbook/templates/DocBook 4` folder.

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 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`.

Author Extensions

The DocBook 5 extensions are the same as the *DocBook 4 extensions*. In addition the table actions work also for HTML tables.

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.

Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp (experimental) and EPUB.

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.

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.

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 install directory.

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 for authoring, producing, and delivering technical information. It divides content into small, self-contained topics that can be reused in different deliverables. The extensibility of DITA permits organizations to define specific information structures and still use standard tools to work with them.

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 Detection*** *option 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 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`.

Author Extensions

The specific actions for DITA topic documents are:

- **Bold** - Surrounds the selected text with a `b` tag.
- **Italic** - Surrounds the selected text with an `i` tag.
- **Underline** - Surrounds the selected text with a `u` tag.

  Note: 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.

- **Cross Reference** - Inserts an `xref` element with the value of attribute `format` set to `dita`. The target of the `xref` is selected in a dialog which lists all the IDs available in a file selected by the user.

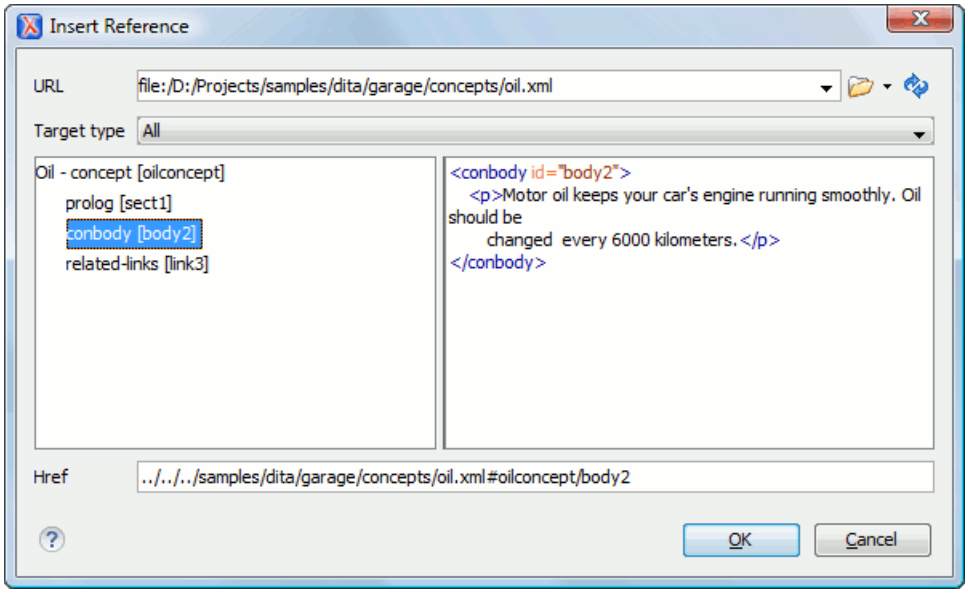


Figure 172: Insert a cross reference in a DITA document

- **Key Reference** - Inserts a user specified element with the value of attribute `keyref` attribute set to a specific key name. As stated in the DITA 1.2 specification keys can be defined at map level which can be then referenced. The target of the `keyref` is selected in a dialog which lists all the keys available in the current opened map from the DITA Maps Manager.

  You can also reference elements at sub-topic level by pressing the **Sub-topic** button and choosing the target.

  Important: 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 page. 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 inside **link** toolbar actions group.

- **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 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.
- **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 Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (the value of `class` attribute of the current element or one of its ancestors contains `topic/p`) then a new paragraph will be inserted after this paragraph. Otherwise a new paragraph is inserted at caret position.
- **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 a graphic object at the caret position. Depending on the current context, an image-type DITA element is inserted. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG. Also you can use this action to *refer MathML files*.
- **Insert 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 will *display 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 *add the catalog to Oxygen*. 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** .

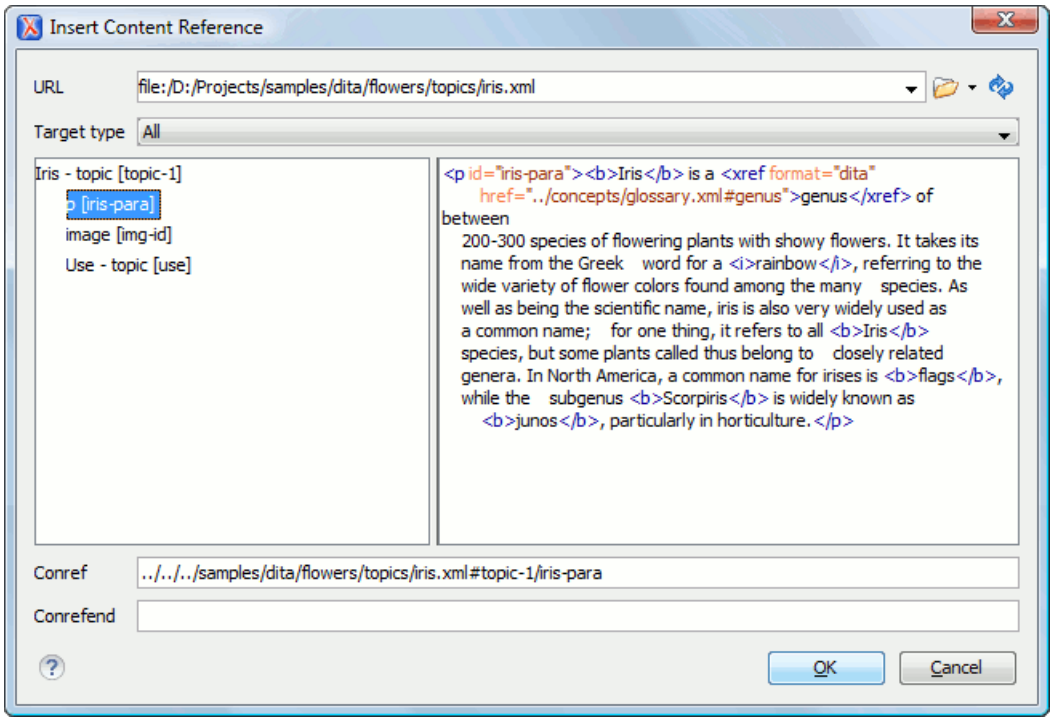


Figure 173: Insert Content Reference Dialog

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 Author 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 will *display 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** .

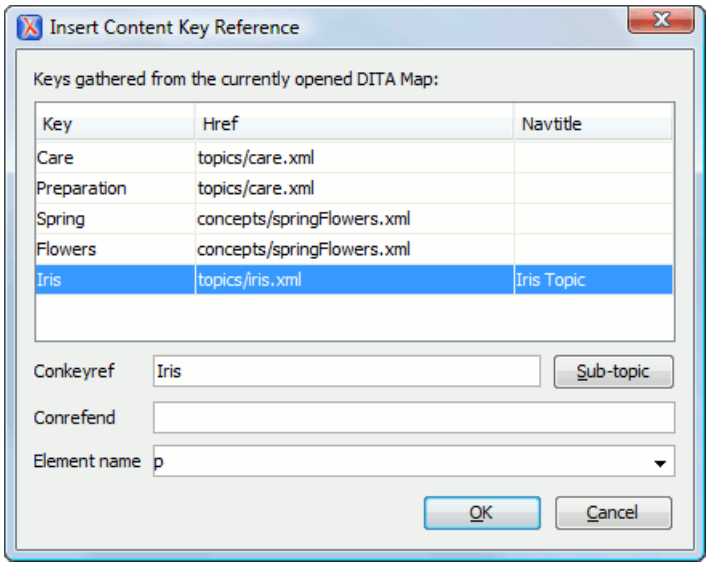


Figure 174: Insert Content Key Reference Dialog

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 Author 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 page.

- **Replace conref / conkeyref reference with content** - Replaces the content reference fragment or the `conkeyref` at caret position with the referenced content. This action is useful when you want to make changes to the content but decide to keep the referenced fragment unchanged.
- **Insert Equation** - Allows you to insert an MathML equation. For more information, see section *Editing MathML Notations*.

- **Create Reusable Component** - Creates a reusable component from a selected fragment of text. For more information, see *Reusing Content*.
- **Insert Reusable Component** - Inserts a reusable component at cursor location. For more information, see *Reusing Content*.
- **Remove Content Reference** - Removes the `conref` attribute of an element. For more information, see *Reusing Content*.
- **Add/Edit Content Reference** - Add or edit the `conref` attribute of an element. For more information, see *Reusing Content*.
- **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 Ordered List** - Inserts an ordered list with one list item.
- **Insert Unordered List** - Inserts an unordered list with one list item.
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure:
  - the number of rows and columns of the table
  - if the header will be generated
  - if the title will be added
  - how the table will be framed
- **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.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **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 cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.

- **Split Cell Below** - splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.

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** - Allows you to generate an unique ID for the element at caret position. You can set the ID pattern using the **ID Generation** dialog, available in the **DITA** main menu, **ID Generation Options** submenu. In this dialog you can specify the elements for which Oxygen generates an ID if the **Auto generate ID's for elements** is enabled. If the element already has an ID, it is preserved.

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 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.

Transformation Scenarios

The following default transformation scenarios are available for DITA Topics:

- **DITA XHTML** - Transforms a DITA topic to XHTML using DITA Open Toolkit 1.5.2;
- **DITA PDF (Idiom FO Plugin)** - Transforms a DITA topic to PDF using the DITA Open Toolkit 1.5.2 and the Apache FOP engine.

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`.

These templates are available when creating *new documents from templates*.

- **DITA - Composite** - New DITA Composite
- **DITA - Concept** - New DITA Concept
- **DITA - General Task** - New DITA Task
- **DITA - Glossentry** - New DITA Glossentry
- **DITA - Reference** - New DITA Reference
- **DITA - Task** - New DITA Task
- **DITA - Topic** - New DITA Topic
- **DITA - Learning Assessment** - New DITA Learning Assessment (learning specialization in DITA 1.2)
- **DITA - Learning Content** - New DITA Learning Content (learning specialization in DITA 1.2)
- **DITA - Learning Summary** - New DITA Learning Summary (learning specialization in DITA 1.2)
- **DITA - Learning Overview** - New DITA Learning Overview (learning specialization in DITA 1.2)

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 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`.

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.

Transformation Scenarios

Predefined transformation scenarios allow you to transform a DITA Map to PDF, XHTML, WebHelp, EPUB and CHM files. Many more output formats are available by clicking the **New** button. The transformation process relies on DITA Open Toolkit 1.5.2.

Templates

The default templates available for DITA maps are stored in `${frameworks}/dita/templates/map` folder. They can be used for easily creating a DITA `map` and `bookmap` files.

These templates are available when creating *new documents from templates*.

- **DITA Map - Bookmap** - New DITA Bookmap
- **DITA Map - Map** - New DITA Map
- **DITA Map - Learning Map** - New DITA learning and training content specialization map
- **DITA Map - Learning Bookmap** - New DITA learning and training content specialization bookmap
- **DITA Map - Eclipse Map** - New DITA learning and training content specialization bookmap

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 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`

Author Extensions

Specific actions are:

- **Bold** - Changes the style of the selected text to `bold` by surrounding it with `b` tag.
- **Italic** - Changes the style of the selected text to `italic` by surrounding it with `i` tag.
- **Underline** - Changes the style of the selected text to `underline` by surrounding it with `u` tag.

  Note: 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.

- **Headings** - Groups actions for inserting `h1`, `h2`, `h3`, `h4`, `h5`, `h6` elements.
- **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `p`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
- **Insert Graphic** - 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 Ordered List** - Inserts an ordered list (`ol` element) with one list item (`li` child element).

- **Insert Unordered List** - Inserts an unordered list (`ul` element) with one list item (`li` child element).
- **Insert Definition List** - Inserts a definition list (`dl` element) with one list item (a `dt` child element and a `dd` child element).
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table, if the header and footer will be generated and how the table will be framed.
- **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.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **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 cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased 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.

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

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.

These templates are 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 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 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`

Author Extensions

The specific actions for TEI P4 documents are:

- **Bold** - Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`.
- **Italic** - Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`.
- **Underline** - Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`.

  Note: 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 Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `p`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
- **Insert Image** - Inserts a graphic object at the caret position. The following dialog is displayed allowing the user to specify the `entity` that refers the image itself.
- **Insert Ordered List** - Inserts an ordered list (`list` element with `type` attribute set to `ordered`) with one list item (`item` element).
- **Insert Itemized List** - Inserts an unordered list (`list` element with `type` attribute set to `bulleted`) with one list item (`item` element).
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table and if the header will be generated.
- **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.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **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 cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Generate IDs** - Allows you to generate ID for the current selection or for the element at caret position if the element appears in **ID Generation** dialog. In this dialog you can specify the elements for which Oxygen should generate an ID. You can choose to automatically generate an ID for these elements by selecting **Auto generate ID's for elements**. You can choose a pattern for the generated ID using the field **ID Pattern**. If the element already has an ID, this ID is preserved.

All actions described above are available in the contextual menu, the **TEI P4** 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 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.

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

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

The TEI P5 Document Type

The TEI P5 document type is the same with that for TEI P4 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 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.

Transformation Scenarios

The following default transformations are available:

- **TEI P5 XHTML** - Transforms a TEI document into a XHTML document
- **TEI P5 PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine

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

The MathML Document Type

Mathematical Markup Language (MathML) is an application of XML for describing mathematical notations and capturing both its structure and content. It aims at integrating mathematical formulae into World Wide Web documents.

Oxygen offers support for editing and validating MathML 2.0 documents.

A file is considered to be a MathML document when the root element name is a `math` or it's namespace is *http://www.w3.org/1998/Math/MathML*.

The schema used for these documents is located in `${frameworks}/mathml2/dtd/mathml2.dtd`, where `${frameworks}` is a subdirectory of the Oxygen install directory.

Templates

There is one default template for MathML stored in the `${frameworks}/mathml2/templates` folder.

This template is available when creating *new documents from templates*.

- **Equation 2.0 DTD Based** - DTD based MathML 2.0 equation template file.
- **Equation 2.0 Schema Based** - XML Schema-based MathML 2.0 equation template file.

The Microsoft Office OOXML Document Type

Office Open XML (also referred to as OOXML or OpenXML) is a free and open *Ecma* international standard document format, and a proposed ISO/IEC standard for representing spreadsheets, charts, presentations and word processing documents.

OOXML uses a file package conforming to the Open Packaging Convention. This format uses the ZIP file format and contains the individual files that form the basis of the document. In addition to Office markup, the package can also include embedded files such as images, videos, or other documents.

Oxygen offers support for editing, transforming and validating documents composing the OOXML package directly through the *archive support*.

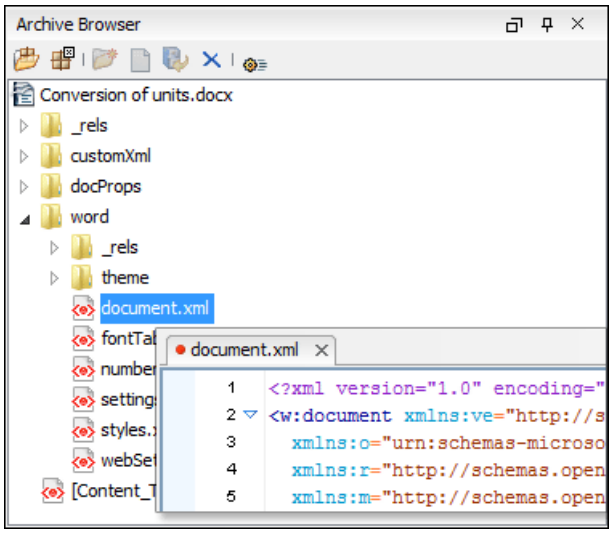


Figure 175: Editing OOXML packages in Oxygen

A file is considered to be an OOXML document when it has one of the following namespaces:

- *http://schemas.openxmlformats.org/wordprocessingml/2006/main*
- *http://schemas.openxmlformats.org/package/2006/content-types*
- *http://schemas.openxmlformats.org/drawingml/2006/main*
- *http://schemas.openxmlformats.org/package/2006/metadata/core-properties*
- *http://schemas.openxmlformats.org/package/2006/relationships*
- *http://schemas.openxmlformats.org/presentationml/2006/main*
- *http://schemas.openxmlformats.org/officeDocument/2006/custom-properties*
- *http://schemas.openxmlformats.org/officeDocument/2006/extended-properties*
- *http://schemas.openxmlformats.org/spreadsheetml/2006/main*
- *http://schemas.openxmlformats.org/drawingml/2006/chart*

The schema used for these documents is located in `${frameworks}/ooxml/schemas/main.nvdl`, where `${frameworks}` is a subdirectory of the Oxygen install directory. The schema can be easily customized to allow user defined extension schemas for use in the OOXML files. See the *Markup Compatibility and Extensibility* Ecma PDF document for more details.

Templates

The default templates for OOXML documents are stored in the `${frameworks}/ooxml/templates` folder. These templates are available when creating *new documents from templates*.

- **OOXML - Microsoft Excel Workbook** - Simple Microsoft Excel XLSX template file
- **OOXML - Microsoft PowerPoint Presentation** - Simple Microsoft PowerPoint PPTX template file
- **OOXML - Microsoft Word Document** - Simple Microsoft Word DOCX template file

The Open Office ODF Document Type

The OpenDocument format (ODF) is a free and open file format for electronic office documents, such as spreadsheets, charts, presentations and word processing documents. The *standard* was developed by the Open Office XML technical committee of the Organization for the Advancement of Structured Information Standards (OASIS) consortium and based on the XML format originally created and implemented by the OpenOffice.org office suite.

A basic OpenDocument file consists of an XML document that has <document> as its root element. OpenDocument files can also take the format of a ZIP compressed archive containing a number of files and directories. These can contain binary content and benefit from ZIP lossless compression to reduce file size. OpenDocument benefits from separation of concerns by separating the content, styles, metadata and application settings into four separate XML files.

Oxygen offers support for editing, manipulating and validating documents composing the ODF package directly through the *archive support*.

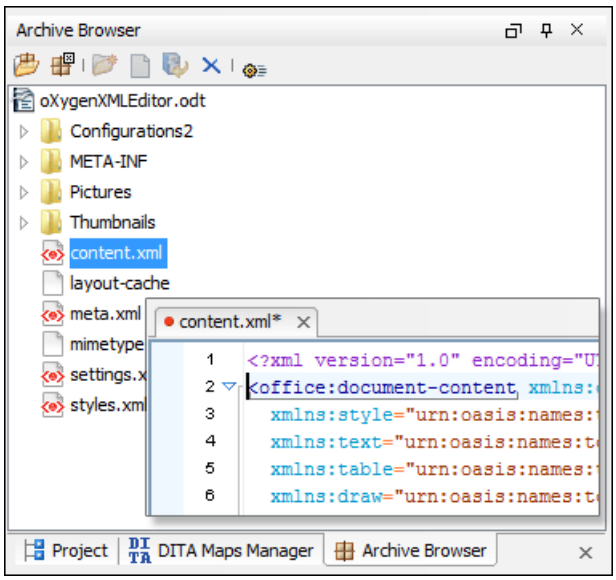


Figure 176: Editing ODF packages in Oxygen

A file is considered to be an ODF document when it has the namespace "urn:oasis:names:tc:opendocument:xmlns:office:1.0".

The schema used for these documents is located in `${frameworks}/odf/schemas/OpenDocument-schema-v1.1.rng`, where `${frameworks}` is a subdirectory of the Oxygen install directory.

Templates

The default templates for ODF documents are stored in the `${frameworks}/odf/templates` folder. These templates are available when creating *new documents from templates*.

- **ODF - Presentation** - Simple Open Office Presentation ODP template file
- **ODF - Spreadsheet** - Simple Open Office Spreadsheet ODS template file
- **ODF - Text Document** - Simple Open Office Text Document ODT template file

The OASIS XML Catalog Document Type

The *OASIS* XML catalog is a document describing a mapping between external entity references or URI's and locally-cached equivalents.

A file is considered to be an XML Catalog document when it has the namespace "urn:oasis:names:tc:entity:xmlns:xml:catalog" or when its root element name is `catalog`.

The OASIS 1.1 schema used for these documents is located in `${frameworks}/xml/catalog1.1.xsd`, where *${frameworks}* is a subdirectory of the Oxygen install directory.

Templates

The default templates for XML catalogs creation are stored in the `${frameworks}/xml/templates` folder. These templates are available when creating *new documents from templates*.

- **OASIS XML Catalog - 1.0** - Sample OASIS 1.0 XML Catalog
- **OASIS XML Catalog - 1.1** - Sample OASIS 1.1 XML Catalog

The XML Schema Document Type

This document type is used to associated CSS stylesheets to an XML Schema so it can be visualized in the Author page.

A file is considered to be an XML Schema document when the root name is `schema` and namespace is *http://www.w3.org/2001/XMLSchema*.

The following CSS alternatives are proposed for visualizing XML Schemas in the Author page.

- `${frameworks}/xmlschema/schema-main.css` - Documentation - representation of XML Schema optimized for editing and viewing documentation
- `${frameworks}/xmlschema/schemaISOSchematron.css` - XMLSchema+ISOSchematron - representation of XML Schema with embedded ISO Schematron rules
- `${frameworks}/xmlschema/schemaSchematron.css` - XMLSchema+Schematron - representation of XML Schema with embedded Schematron rules
- `${frameworks}/xmlschema/default.css` - representation of XML Schema for general editing

The Relax NG Document Type

This document type is used to associated CSS stylesheets to an Relax NG file so it can be visualized in the Author page.

A file is considered to be an Relax NG document when the namespace is *http://relaxng.org/ns/structure/1.0*.

The following CSS alternatives are proposed for visualizing RelaxNG schemas in the Author page.

- `${frameworks}/relaxng/relaxng-main.css` - Relax NG - representation of Relax NG optimized for editing in the Author mode
- `${frameworks}/relaxng/relaxngISOSchematron.css` - Relax NG (XML Syntax)+ISOSchematron - representation of Relax NG (XML syntax) with embedded ISO Schematron rules. Embedded Schematron rules are not supported in Relax NG schemas with compact syntax.
- `${frameworks}/relaxng/relaxngSchematron.css` - Relax NG (XML Syntax)+Schematron - representation of Relax NG (XML syntax) with embedded Schematron rules. Embedded Schematron rules are not supported in Relax NG schemas with compact syntax.

The NVDL Document Type

This document type is used to associated CSS stylesheets to a NVDL file so it can be visualized in the Author page.

A file is considered to be a NVDL document when the namespace is *http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0*.

The following CSS is proposed for visualizing NVDL schemas in the Author page:

- `${frameworks}/nvdl/nvdl.css` - Representation of Relax NG schema optimized for editing in the Author mode.

The Schematron Document Type

This document type is used to associate CSS stylesheets to a Schematron file so it can be visualized in the Author page.

A file is considered to be a Schematron document when the namespace is *http://purl.oclc.org/dsdl/schematron*.

The following CSS is proposed for visualizing Schematron schemas in the Author page:

- `${frameworks}/schematron/iso-schematron.css` - Representation of Schematron optimized for editing in the Author mode.

The Schematron 1.5 Document Type

This document type is used to associate CSS stylesheets to a Schematron 1.5 file so it can be visualized in the Author page.

A file is considered to be a Schematron 1.5 document when the namespace is *http://www.ascc.net/xml/schematron*.

The following CSS is proposed for visualizing Schematron 1.5 schemas in the Author page:

- `${frameworks}/schematron/schematron15.css` - Representation of Schematron 1.5 optimized for editing in the Author mode.

The XSLT Document Type

This document type is used to associate CSS stylesheets to an XSLT stylesheet file so it can be visualized in the Author page.

A file is considered to be a XSLT document when the namespace is *http://www.w3.org/1999/XSL/Transform*.

The following CSS is proposed for visualizing XSLT stylesheets in the Author page:

- `${frameworks}/xslt/xslt.css` - Representation of XSLT optimized for editing in the Author mode.

The XMLSpec Document Type

XMLSpec is a markup language for W3C specifications and other technical reports.

A file is considered to be an XMLSpec document when the root name is `spec`.

XMLSpec documents use a Relax NG schema located in `${frameworks}/xmlspec/schema/xmlspec.rng`, where `${frameworks}` is a subdirectory of the Oxygen install directory.

The default XML catalog is stored in `${frameworks}/xmlspec/catalog.xml`.

Transformation Scenarios

The following default transformation scenarios are available:

- **XMLSpec PDF** - Transforms an XMLSpec document into PDF document using the Apache FOP engine
- **XMLSpec HTML** - Transforms an XMLSpec document into HTML document
- **XMLSpec HTML Diff** - Produces color-coded HTML from *diff* markup
- **XMLSpec HTML Slices** - Produces chunked HTML specifications

Templates

The default templates for XMLSpec are stored in `${frameworks}/xmlspec/templates` folder and they can be used for easily creating an XMLSpec. These templates are available when creating *new documents from templates*.

- **XMLSpec - New Document** - New XMLSpec document

The FO Document Type

FO describes the formatting of XML data for output to screen, paper or other media.

A file is considered to be an FO document when the namespace is *http://www.w3.org/1999/XSL/Format*.

FO documents use a XML Schema located in `${frameworks}/fo/xsd/fo.xsd`, where `${frameworks}` is a subdirectory of the Oxygen install directory.

Transformation Scenarios

The following default transformation scenarios are available:

- **FO PDF** - Transforms an FO document into PDF document using the Apache FOP engine.

The EAD Document Type

EAD Document Type Definition (DTD) is a standard for encoding archival finding aids using Extensible Markup Language (XML). The standard is maintained in the Network Development and MARC Standards Office of the Library of Congress (LC) in partnership with the Society of American Archivists.

A file is considered to be an EAD document when its namespace is “urn:isbn:1-931666-22-9” or its public ID is “//DTD ead.dtd (Encoded Archival Description (EAD) Version 2002)//EN”.

EAD documents use a Relax NG Schema located in `${frameworks}/ead/rng/ead.rng`, where `${frameworks}` is a subdirectory of the Oxygen install directory.

The default XML catalog is stored in `${frameworks}/ead/catalog.xml`.

Templates

The default templates for EAD are stored in `${frameworks}/ead/templates` folder and they can be used for easily creating an EAD document. These templates are available when creating *new documents from templates*.

- **EAD - NWDA Template 2008-04-08** - New EAD document

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.

Chapter 8

Author Developer Guide

Topics:

- *Simple Customization Tutorial*
- *Advanced Customization Tutorial - Document Type Associations*
- *CSS Support in Oxygen Author*
- *Example Files Listings - The Simple Documentation Framework Files*
- *Oxygen XML Author Component*

The Author editor from Oxygen 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 user manual and as a separate document in *the Author SDK*.

Figure 177: Oxygen Author Visual Editor

Although Oxygen 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 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 Oxygen 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 standout 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;
}
```

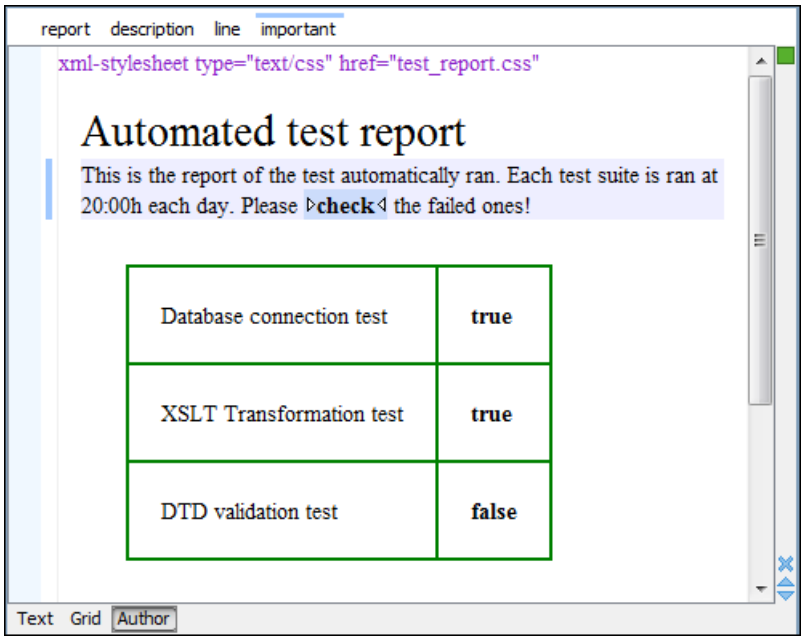


Figure 178: A report opened in the Author

The XML Instance Template

Based on the XML Schema and the CSS file the Oxygen 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 Author 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*.

In this tutorial a **Document Type Association** will be created for a set of documents. As an example a light documentation framework (similar to DocBook) will be created, then complete customization of the Author editor will be set up.

👉 **Note:** The samples used in this tutorial can be found in the *Example Files Listings*.

👉 **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 website*.

👉 **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen website*. Also it can be downloaded as a *zip archive from the website*.

Author 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.

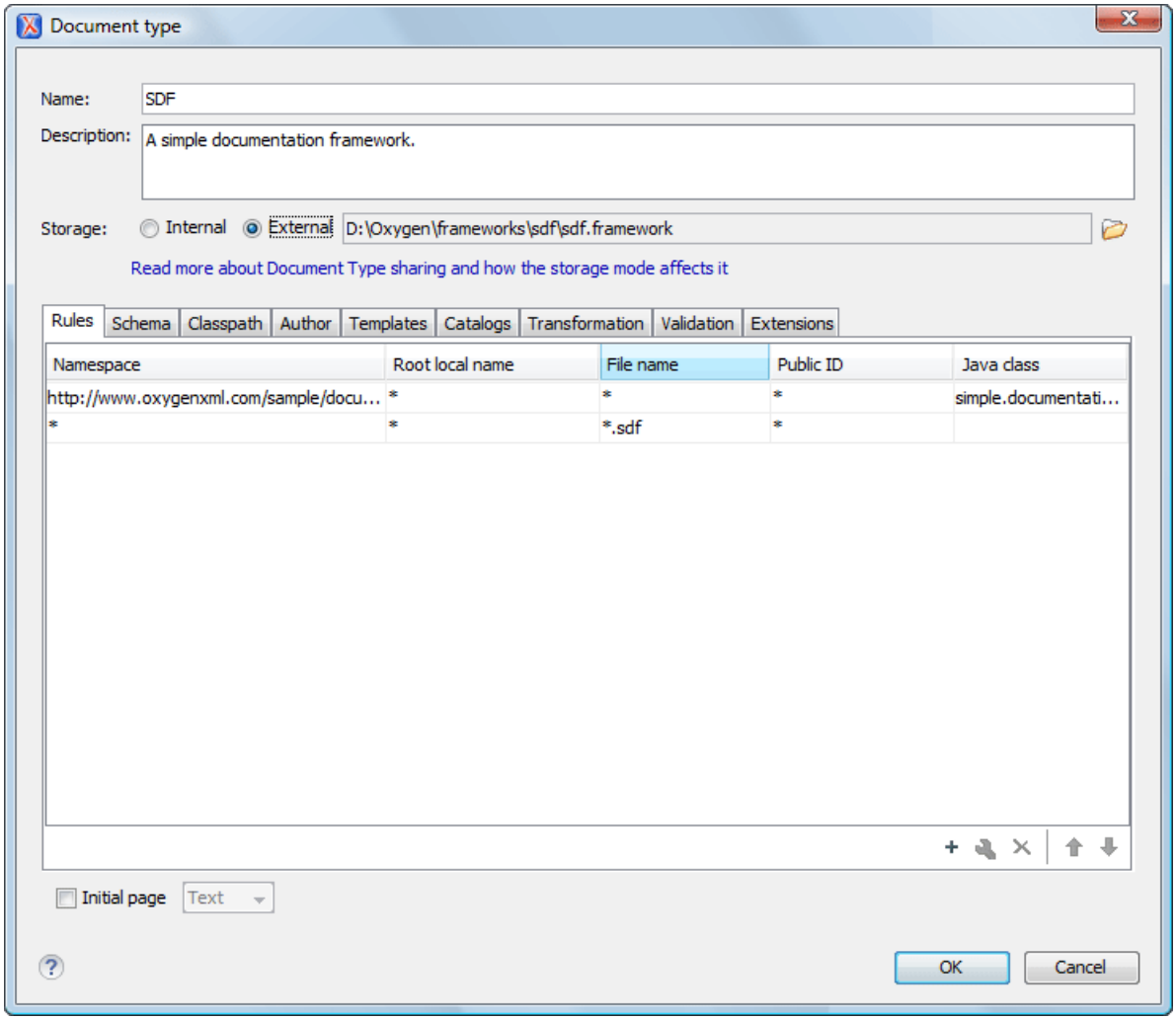


Figure 179: The Document Type Dialog

You can specify the following properties for a document type:

- **Name** - The name of the document type.
- **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 project file for Project-level Document Type Association Options or in the Oxygen 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 page** - Allows you to select the initial editing mode (**Text**, **Author**, **Grid**, **Design** - available only for the W3C XML Schema editor) for this document type.

Note: The initial page for an document type can also be customized from the **Pages** preferences panel located in **Options** > **Preferences** > **Editor** > **Pages**

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.
- **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

The Oxygen Author toolbars and menus can be changed to provide a productive editing experience for the content authors. You can create a set of actions that are specific to a document type.

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. Open the **Options Dialog**, and select the **Document Types Association** option pane.
2. In the lower part of the **Document Type Association** dialog, click on the **Author** tab, then select the **Actions** label.
3. To add a new action click on the + **Add** button.

The Insert Section Action

This section shows all the steps needed 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.

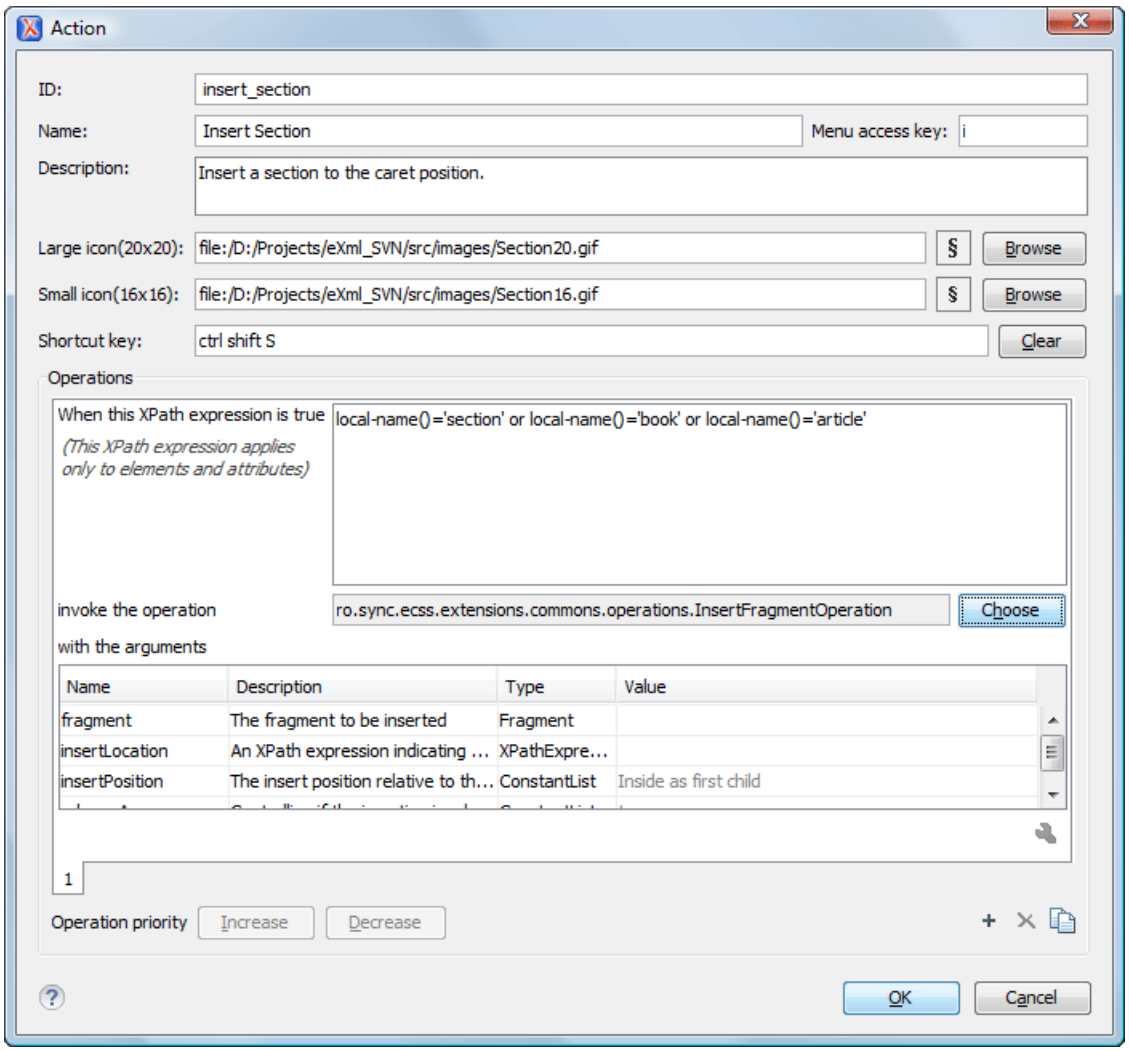


Figure 180: The Action Edit Dialog

1. Set the **ID** field to **insert_section**. This is an unique action identifier.
2. Set the **Name** field to **Insert Section**. This will be the action's name, displayed as a tooltip when the action is placed in the toolbar, or as the menu item name.
3. Set the **Menu access key** to **i**. On Windows, the menu items can be accessed using (ALT + letter) 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+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 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+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()]
   ```

insertPosition - Select **After**.

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.

The first thing to check is that the toolbar **Author custom actions 1** is displayed when switching to the **Author** mode: right click in the upper part of application window, in the area that contains the toolbar buttons and check if **Author custom actions 1** in the displayed menu if it is unchecked.

1. Open the Document Type edit dialog for the **SDF** framework and select on the **Author** tab. Next click on the **Toolbar** label.

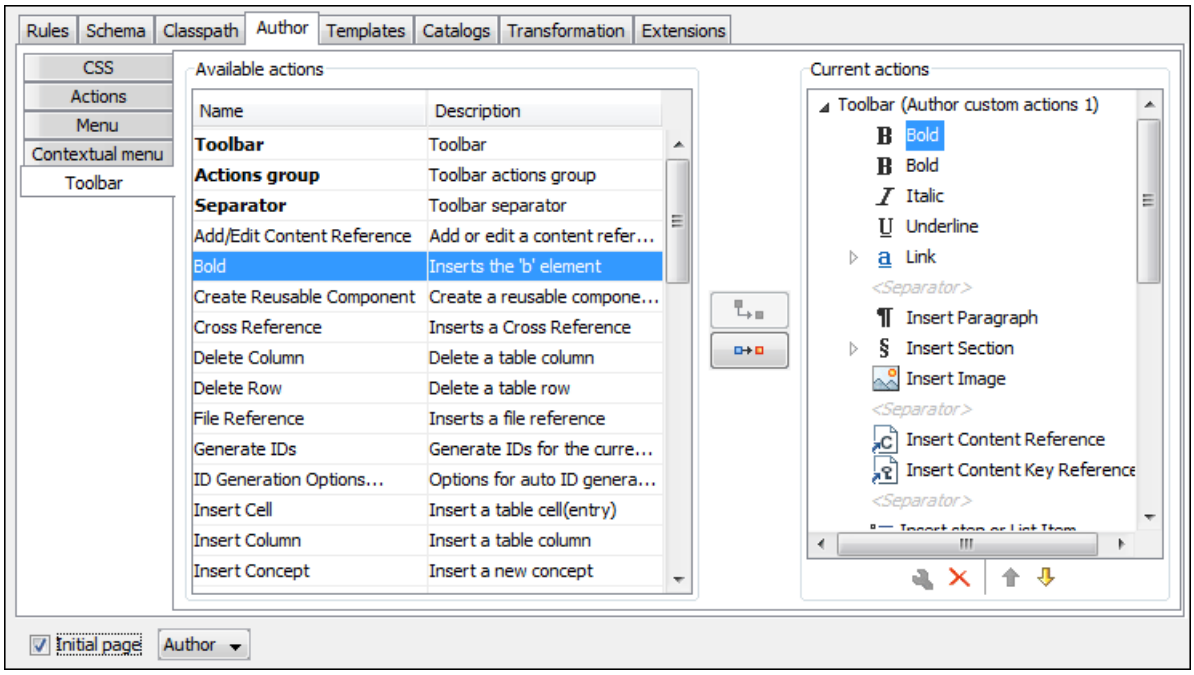
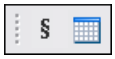


Figure 181: Configuring the Toolbar

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 182: Author Custom Actions Toolbar



Tip: If you have many custom toolbar actions or want to group actions according to their category you can add additional toolbars with custom names and split the actions to better suit your purpose.

Configuring the Main Menu

Defined actions can be grouped into customized menus in the Oxygen 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.

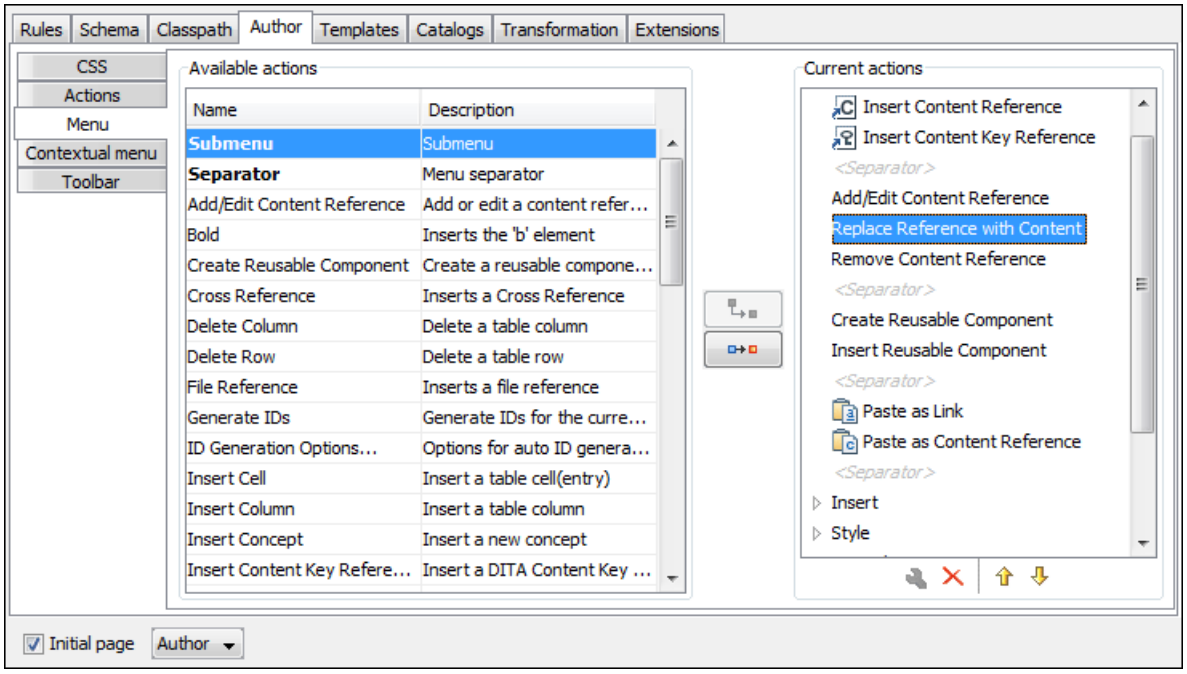


Figure 183: Configuring the Menu

When opening a **Simple Documentation Framework** test document in Author mode, the menu you created is displayed in the editor menu bar, between the Debugger 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).

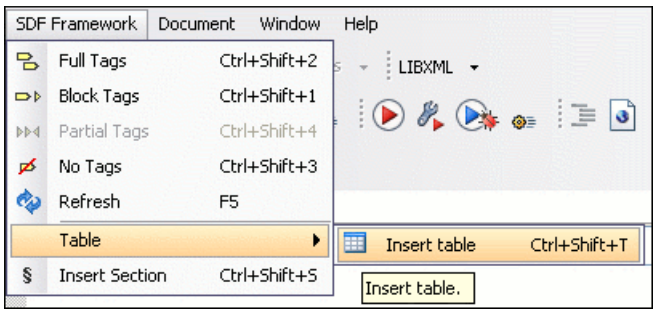


Figure 184: Author Menu

Configuring the Contextual Menu

The contextual menu is shown when you right click (**ctrl + 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.

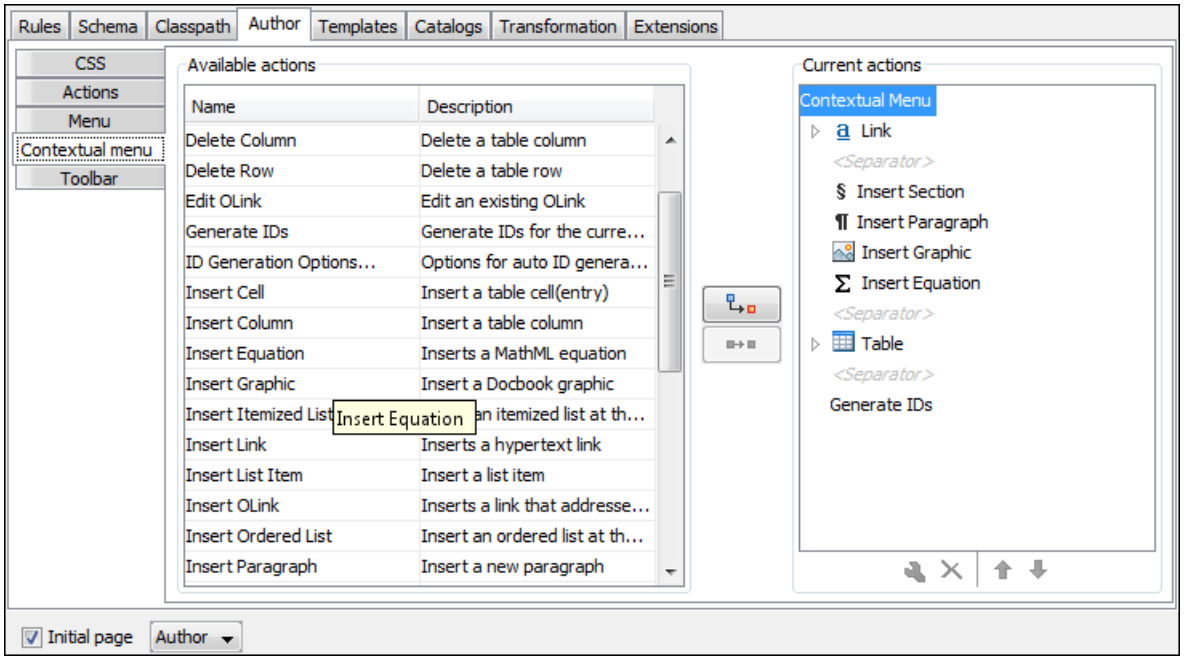


Figure 185: Configuring the Contextual Menu

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.

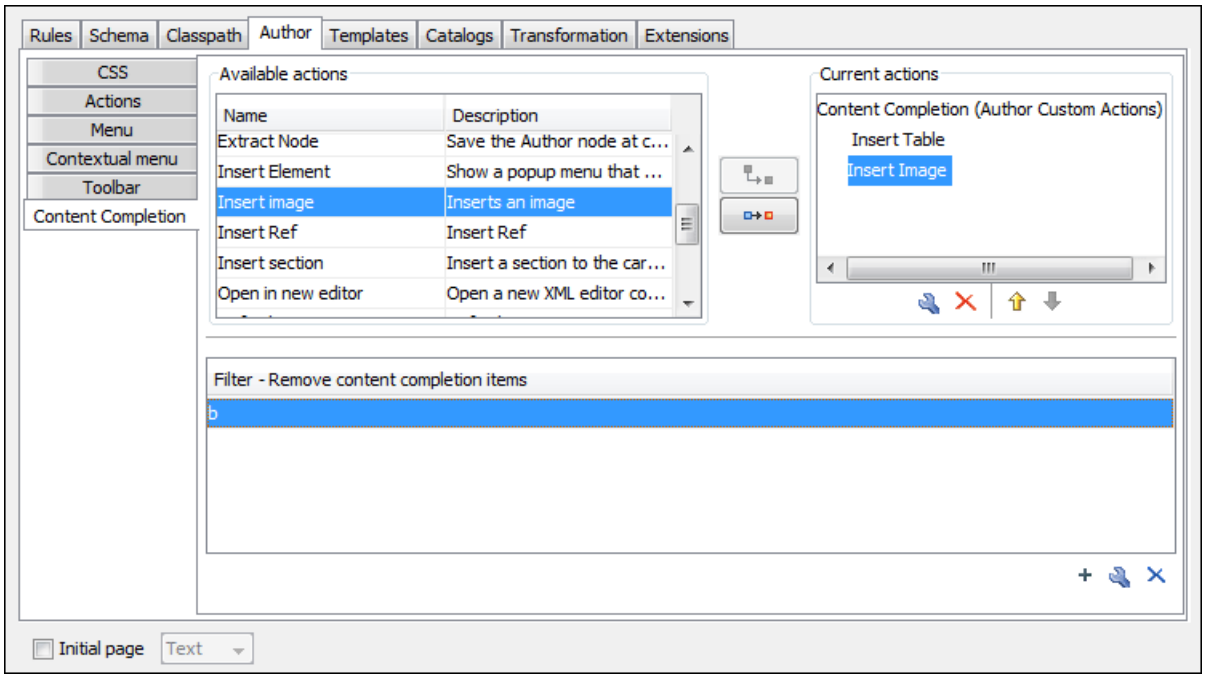


Figure 186: Customize Content Completion

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** items lists. The bottom side contains the list with all the items that you decided to remove from the **Content Completion** items lists.

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:

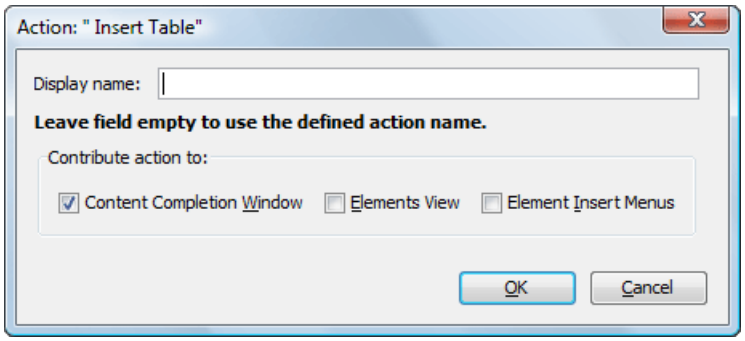


Figure 187: Insert action dialog

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.

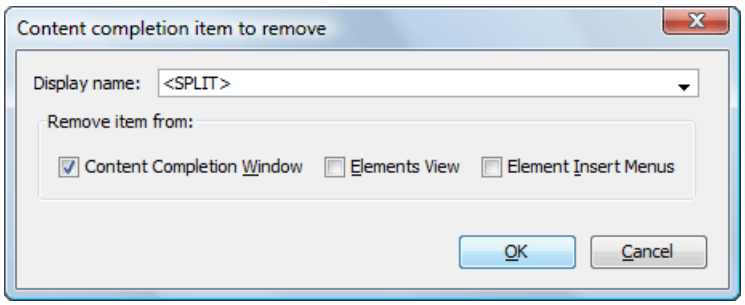


Figure 188: Remove item dialog

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.

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.

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.

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 will be placed before the selection.

footer — The text that will be placed after the selection.

The arguments of `InsertFragmentOperation` operation

fragment — The value for this argument is a text. This is parsed by Oxygen Author 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.

Note:

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!

Note:

If the fragment contains namespace declarations that are identical to those found in the document, the namespace declaration attributes will be removed from elements contained by the inserted fragment.

There are two possible scenarios:

1. **Prefixes that are not bound explicitly**

   For instance, the fragment:

   ```
   <x:item id="dty2"/>
   &ent;
   <x:item id="dty3"/>
   ```

   Can be correctly inserted in the document: ('|' marks the insertion point):

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE x:root [
       <!ENTITY ent "entity">
   ]>

   <x:root xmlns:x="nsp">
     |
   </x:root>
   ```

   Result:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE x:root [
       <!ENTITY ent "entity">
   ]>
   <x:root xmlns:x="nsp">
       <x:item id="dty2"/>
       &ent;
       <x:item id="dty3"/>
   </x:root>
   ```

2. **Default namespaces**

If there is a default namespace declared in the document and the document fragment does not declare a namespace, the elements from the fragment are considered to be in **no namespace**.

For instance the fragment:

```
<item id="dty2"/>
<item id="dty3"/>
```

Inserted in the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
|
</root>
```

Gives the result document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
    <item xmlns="" id="dty2"/>
```

```
    <item xmlns="" id="dty3"/>
</root>
```

insertLocation An XPath expression that is relative to the current node. It selects the reference node for the fragment insertion.

insertPosition One of the three constants: "**Inside**", "**After**", or "**Before**" , showing where the insertion is made relative to the reference node selected by the **insertLocation**. "**Inside**" has the meaning of the first child of the reference node.

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 Action 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 page 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 set the user role to Developer (you need write access to the `OXYGEN_INSTALLATION_DIR`). Edit the document type.
   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 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 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 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 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 OxygenXML Editor . Otherwise the classes may not be loaded by the Java virtual machine. For example if you run OxygenXML Editor with a Java 1.5 virtual machine but the Java classes of your custom Author operations are compiled with a Java 1.6 virtual machine then the custom operations cannot be loaded and used by the Java 1.5 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 `AuthorAccess` and a map
- argument names and values

- The `getArguments` method is used by Oxygen 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 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 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'/>
```

3. 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>
```
4. Copy the `sdf.jar` file into the `frameworks / sdf` folder.
5. Add the `sdf.jar` to the Author class path. To do this, open the **Options** > **Preferences** > **Document Type Association** dialog, select **SDF** and press the **Edit** button.
6. 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 `frameworks` folder.
7. Let's create now the action which will use the defined operation. Click on the **Actions** label. Copy the icon files for the menu item and for the toolbar in the `frameworks / sdf` folder.
8. Define the action's properties:
   - Set **ID** to **insert_image**.
   - Set **Name** to **Insert image**.
   - Set **Menu access key** to letter **i**.
   - Set **Toolbar action** to **${frameworks}/sdf/toolbarImage.png**.
   - Set **Menu icon** to **${frameworks}/sdf/menuImage.png**.
   - Set **Shortcut key** to **Ctrl+Shift+i**.
9. Now let's set up the operation. You want to add images only if the current element is a `section`, `book` or `article`.
   - Set the value of **XPath expression** to
     ```
     local-name()='section' or local-name()='book'
      or local-name()='article'
     ```
   - Set the **Invoke operation** field to `simple.documentation.framework.InsertImageOperation`.

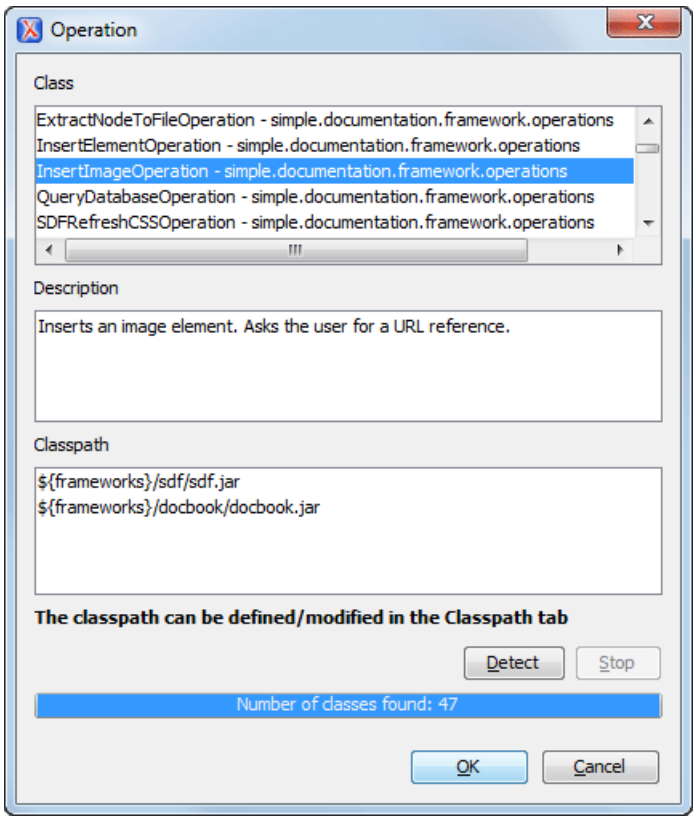


Figure 189: Selecting the Operation

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 implements the `ro.sync.ecss.extensions.api.AuthorOperation` interface.

```
import ro.sync.ecss.extensions.api.ArgumentDescriptor;
import ro.sync.ecss.extensions.api.ArgumentsMap;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperation;
import ro.sync.ecss.extensions.api.AuthorOperationException;

public class QueryDatabaseOperation implements AuthorOperation{
```

3. Now define the operation's arguments. For each of them you will use a `String` constant representing the argument name:

```
private static final String ARG_JDBC_DRIVER ="jdbc_driver";
private static final String ARG_USER ="user";
private static final String ARG_PASSWORD ="password";
```

```
private static final String ARG_SQL ="sql";
private static final String ARG_CONNECTION ="connection";
```

4. You must describe each of the argument name and type. To do this implement the `getArguments` method which will return an array of argument descriptors:

```
public ArgumentDescriptor[] getArguments() {
  ArgumentDescriptor args[] = new ArgumentDescriptor[] {
    new ArgumentDescriptor(
      ARG_JDBC_DRIVER,
      ArgumentDescriptor.TYPE_STRING,
      "The name of the Java class that is the JDBC driver."),
    new ArgumentDescriptor(
      ARG_CONNECTION,
      ArgumentDescriptor.TYPE_STRING,
      "The database URL connection string."),
    new ArgumentDescriptor(
      ARG_USER,
      ArgumentDescriptor.TYPE_STRING,
      "The name of the database user."),
    new ArgumentDescriptor(
      ARG_PASSWORD,
      ArgumentDescriptor.TYPE_STRING,
      "The database password."),
    new ArgumentDescriptor(
      ARG_SQL,
      ArgumentDescriptor.TYPE_STRING,
      "The SQL statement to be executed.")
  };
  return args;
 }
```

   These names, types and descriptions will be listed in the **Arguments** table when the operation is configured.

5. When the operation is invoked, the implementation of the `doOperation` method extracts the arguments, forwards them to the method that connects to the database and generates the XML fragment. The XML fragment is then inserted at the caret position.

```
public void doOperation(AuthorAccess authorAccess, ArgumentsMap map)
   throws IllegalArgumentException, AuthorOperationException {

  // Collects the arguments.
  String jdbcDriver =
   (String)map.getArgumentValue(ARG_JDBC_DRIVER);
  String connection =
   (String)map.getArgumentValue(ARG_CONNECTION);
  String user =
   (String)map.getArgumentValue(ARG_USER);
  String password =
   (String)map.getArgumentValue(ARG_PASSWORD);
  String sql =
   (String)map.getArgumentValue(ARG_SQL);

  int caretPosition = authorAccess.getCaretOffset();
  try {
   authorAccess.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 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+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.

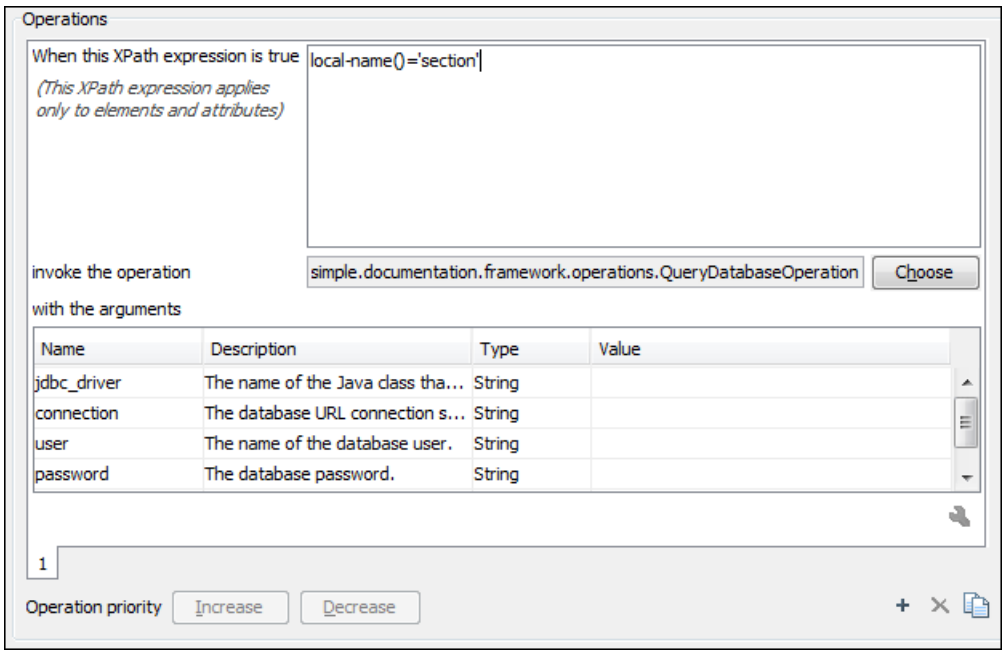


Figure 190: Java Operation Arguments Setup

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.

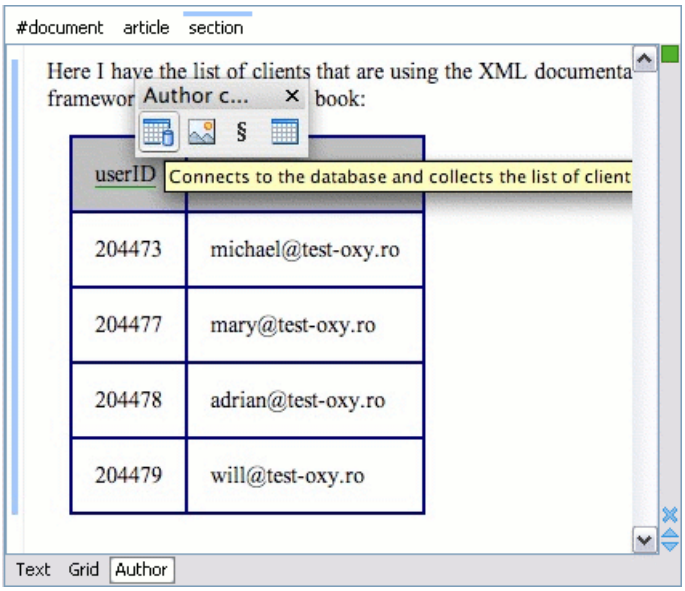


Figure 191: Table Content Extracted from the Database

Configuring New File Templates

You will create a set of document templates that the content authors will use as starting points for creating new *Simple Document Framework* books and articles.

Each of the Document Type Associations 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. Create the templates directory into the `frameworks / sdf` directory. The directory tree for the documentation framework is now:

```
oxygen
    frameworks
        sdf
            schema
            css
            templates
```

2. Now let's create in this `templates` directory two files, one for the *book* template and another for the *article* template.

   The `Book.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>Book Template Title</title>
    <section>
        <title>Section Title</title>
        <abs:def/>
        <para>This content is copyrighted:</para>
        <table>
            <header>
                <td>Company</td>
                <td>Date</td>
            </header>
            <tr>
                <td/>
                <td/>
            </tr>
        </table>
    </section>
    </book>
```

   The `Article.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<article
    xmlns="http://www.oxygenxml.com/sample/documentation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <title></title>
    <section>
        <title></title>
        <para></para>
        <para></para>
    </section>
</article>
```

   You can also use *editor variables* in the template files' content and they will be expanded when the files are opened.

3. Open the Document Type dialog for the **SDF** framework and click on the **Templates** tab. Enter in the **Templates directory** text field the value `${frameworksDir} / sdf / templates`. As you already seen before, it is recommended that all the file references made from a Document Type Association to be relative to the `${frameworksDir}` 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, press the **File**/**New** menu item to display the **New** dialog. The names of the two templates are prefixed with the name of the Document Type Association, in our case **SDF**. Selecting one of them should create a new XML file with the content specified in the template file.

Configuring XML Catalogs

In the XML sample file for **SDF** you did not use a xsi:schemaLocation attribute, but instead you let the editor use the schema from the association. However there are cases in which you must 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">
       <system systemId="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 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:
>
> ```
> xsi:schemaLocation="http://www.oxygenxml.com/sample/documentation/abstracts
>
>     http://www.oxygenxml.com/SDF/abs.xsd"/>
> ```
>
> The catalog mapping is:
>
> ```
> http://www.oxygenxml.com/SDF/abs.xsd -> schema/abs.xsd
> ```
>
> This means that all the references to http://www.oxygenxml.com/SDF/abs.xsd must be resolved to the `abs.xsd` file located in the `schema` directory. The URI element is used by URI resolvers, for example for resolving a URI reference used in an XSLT stylesheet.

Configuring Transformation Scenarios

When distributing a framework to the users, it is a good idea to have the transformation scenarios already configured. This 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*.

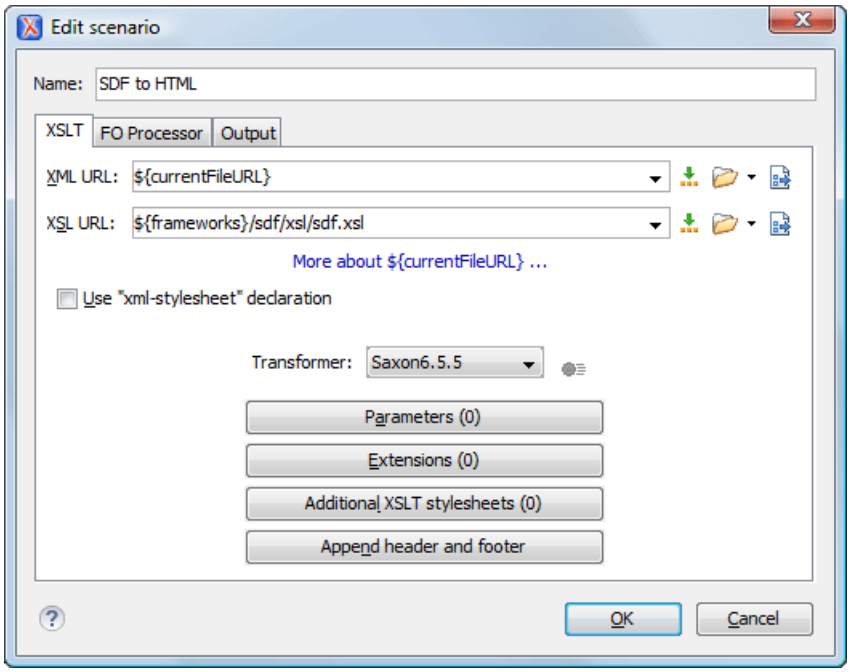


Figure 192: Configuring a transformation scenario

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** option.
   - Enable the **Saved file** option.

Now the scenario is listed in the **Transformation** tab:

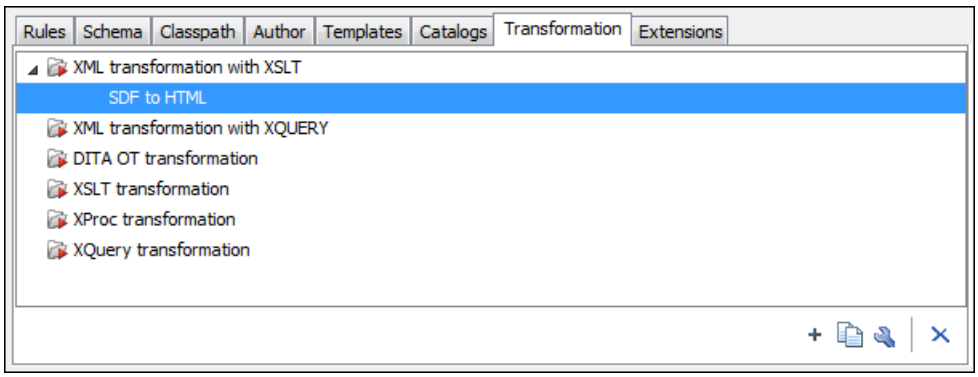


Figure 193: The transformation tab

To test the transformation scenario you just created, open the **SDF** XML sample from the *Example Files Listings*. Click on the **Apply Transformation Scenario** button to display the **Configure Transformation 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.

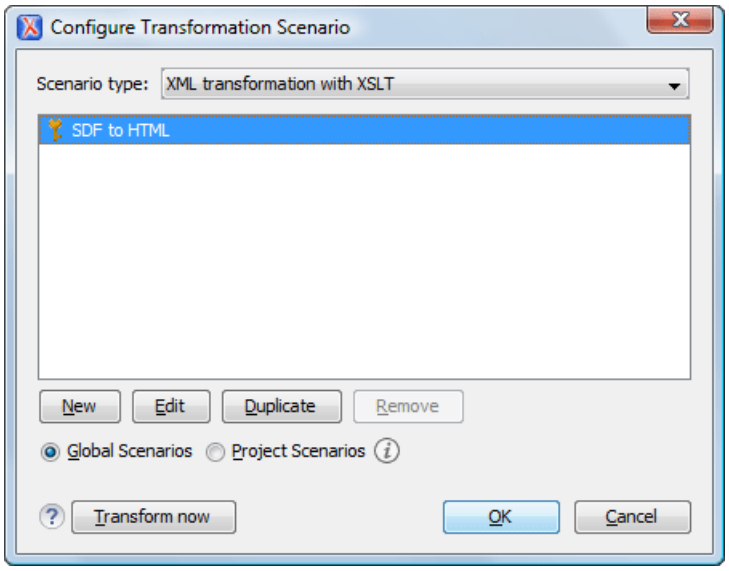


Figure 194: Selecting the predefined scenario

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.

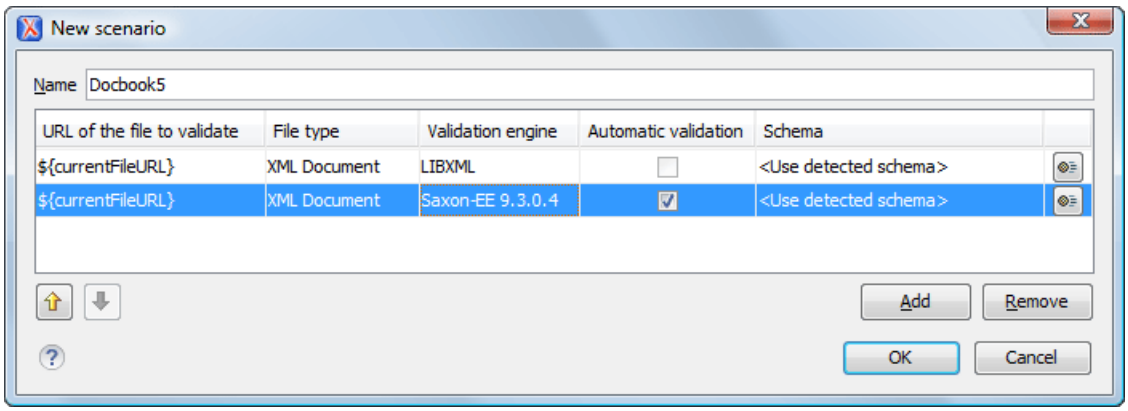


Figure 195: Add / Edit a Validation Unit

The table holds the following information:

- **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 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 for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the validation is done by the default engine set in Preferences pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, etc) 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** - Active when you set the **File type** to **XML Document**.
- **Settings** - Contains an action that allows you to set a schema, when validating XML documents, or a list of extensions when validating XSL or XQuery documents.

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:

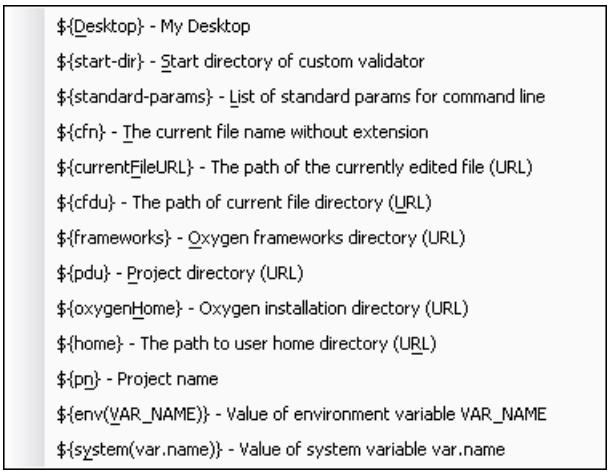


Figure 196: Insert an Editor Variable

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*.
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.

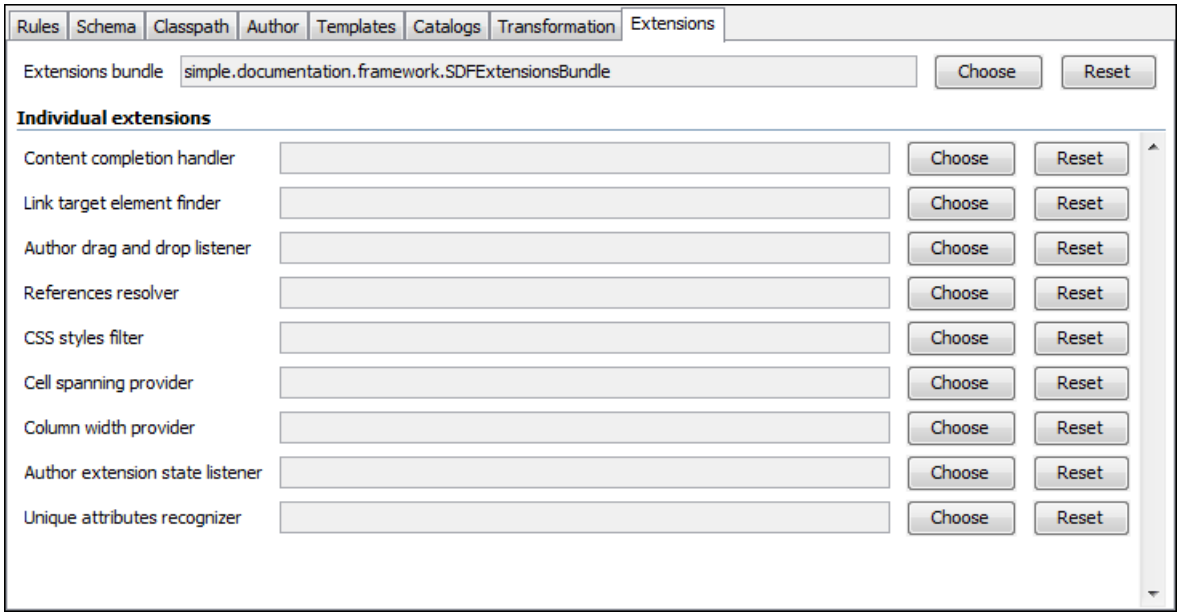


Figure 197: Configure extensions for a document type

Configuring an Extensions Bundle

Starting with Oxygen 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.

The extensions bundle is represented by the `ro.sync.ecss.extensions.api.ExtensionsBundle` class. The provided implementation of the `ExtensionsBundle` is instantiated when the rules of the Document Type Association defined for the custom framework match a document opened in the editor. Therefor references to objects which need to be persistent throughout the application running session must not be kept in the bundle because the next detection event can result in creating another `ExtensionsBundle` instance.

👉 **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen 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 page. The listener is notified about the deactivation when another framework is activated for the same document, the user switches to another page 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 `AuthorSchemaAwareEditingHandler`. The handler can either resolve a specific case, let the default implementation take place or reject the edit entirely by throwing an `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 SDFSchemaManagerFilter();
}
```

   A detailed presentation of the schema manager filter can be found in *Configuring a Content completion handler* section.

6. The Oxygen 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 page, 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 page for both Oxygen Eclipse plugin and standalone application. The Text page corresponding listener is available only for Oxygen Eclipse plugin. The methods corresponding to each implementation are: `createAuthorAWTDndListener`, `createTextSWTDndListener` and `createAuthorSWTDndListener`.

```
public AuthorDnDListener createAuthorAWTDndListener() {
     return new SDFAuthorDndListener();
}
```

For more details about the Author drag and drop listeners see the *Configuring a custom Drag and Drop listener* section.

8. Another extension which can be included in the bundle is the reference resolver. In our case the references are represented by the **ref** element and the attribute indicating the 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 page matches the Document Type Association where the extensions bundle is defined. The instantiated references resolver object is kept and used until another extensions bundle corresponding to another Document Type is activated as result of the detection process.

```
public AuthorReferenceResolver createAuthorReferenceResolver() {
     return new ReferencesResolver();
}
```

A more detailed description of the references resolver can be found in the *Configuring a References Resolver* section.

9. To be able to dynamically customize the default CSS styles for a certain `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 page matches the document type association where the extensions bundle is defined. The instantiated filter object is kept and used until another extensions bundle corresponding to another Document Type is activated as a result of the detection process.

```
public StylesFilter createAuthorStylesFilter() {
     return new SDFStylesFilter();
}
```

See the *Configuring CSS styles filter* section for more details about the styles filter extension.

10. In order to edit data in custom tabular format implementations of the `ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider` and `the ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` interfaces should be provided. The two methods from the `ExtensionsBundle` specifying these two extension points are `createAuthorTableCellSpanProvider` and `createAuthorTableColumnWidthProvider`.

```
public AuthorTableCellSpanProvider createAuthorTableCellSpanProvider() {
     return new TableCellSpanProvider();
}

public AuthorTableColumnWidthProvider
```

```
          createAuthorTableColumnWidthProvider() {
      return new TableColumnWidthProvider();
}
```

The two table information providers are not reused for different tables. The methods are called for each table in the document so new instances should be provided every time. Read more about the cell span and column width information providers in *Configuring a Table Cell Span Provider* and *Configuring a Table Column Width Provider* sections.

If the functionality related to one of the previous extension point does not need to be modified then the developed `ExtensionsBundle` should not override the corresponding method and leave the default base implementation to return **null**.

11. Package the compiled class into a jar file.
12. Copy the jar file into the `frameworks / sdf` directory.
13. Add the jar file to the Author class path.
14. 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 website*.

Preserve Style Markup 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, etc. 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 markup but it can be customized by setting an XSLT stylesheet in that document type. The XSLT stylesheet should 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 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 page, 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 `OptionsStorage` can be obtained by calling the `getOptionsStorage` method from the author access. The same object can be used to register `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 `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 `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 page.

The **deactivation** event is received when another framework is activated for the same document, the user switches to another editor page 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

You can implement your own handler for actions like typing, delete or paste by providing an implementation of `ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler`. The *Schema Aware Editing* must be **On** or **Custom** in order for this handler to be called. 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 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().isEnableSmartTyping())
{
    try {
      AuthorDocumentFragment characterFragment =

authorAccess.getDocumentController().createNewDocumentTextFragment(String.valueOf(ch));

      handleTyping = handleInsertionEvent(offset, new AuthorDocumentFragment[]
{characterFragment}, authorAccess);
    } catch (AuthorOperationException e) {
```

```
      throw new InvalidEditException(e.getMessage(), "Invalid typing event: " +
 e.getMessage(), e, false);
    }
  }
  return handleTyping;
}
```

Implementing the `AuthorSchemaAwareEditingHandler` gives the possibility to handle other events like: the keyboard delete event at the given offset (using Delete or Backspace keys), delete element tags, delete selection, join elements or paste fragment.

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen 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 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 SDFSchemaManagerFilter implements SchemaManagerFilter {
```

You can implement the various callbacks of the interface either by returning the default values given by Oxygen 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 `SDFSchemaManagerFilter` checks if the element from the current context is the `table` element and adds the `frame` attribute to the `table` list of attributes.

```
/**
 * Filter attributes of the "table" element.
 */
public List<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 `SDFSchemaManagerFilter` uses this method to replace the `td` child element with the `th` element when `header` is the current context element.

```
public List<CIElement> filterElements(List<CIElement> elements,
    WhatElementsCanGoHereContext context) {
  // If the element from the current context is the 'header' element remove the

  // 'td' element from the list of content completion proposals and add the
  // 'th' element.
  if (context != null) {
    Stack<ContextElement> elementStack = context.getElementStack();
    if (elementStack != null) {
      ContextElement contextElement = context.getElementStack().peek();
      if ("header".equals(contextElement.getQName())) {
        if (elements != null) {
          for (Iterator<CIElement> iterator = elements.iterator();
iterator.hasNext();) {
            CIElement element = iterator.next();
            // Remove the 'td' element
            if ("td".equals(element.getQName())) {
              elements.remove(element);
              break;
            }
          }
        } else {
          elements = new ArrayList<CIElement>();
        }
        // Insert the 'th' element in the list of content completion proposals
        CIElement thElement = new SDFElement();
        thElement.setName("th");
        elements.add(thElement);
      }
    }
  } else {
    // If the given context is null then the given list of content completion
elements contains
    // global elements.
  }
  return elements;
}
```

The elements or attributes values can be filtered using the `filterElementValues` or `filterAttributeValues` methods.

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen 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 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 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 *NCName* identifies an element by means of stepwise navigation, starting from the element located by the given name.

The constructor separates the id/integers which are delimited by slashes(/) into a sequence of identifiers (an XPointer path). It will also check that the link has one of the supported patterns of the XPointer element() scheme.

```
public XPointerElementLocator(IDTypeVerifier idVerifier, String link)
                      throws ElementLocatorException {
```

```
  super(link);
  this.idVerifier = idVerifier;

  link = link.substring("element(".length(), link.length() - 1);

  StringTokenizer stringTokenizer = new StringTokenizer(link, "/", false);
  xpointerPath = new String[stringTokenizer.countTokens()];
  int i = 0;
  while (stringTokenizer.hasMoreTokens()) {
    xpointerPath[i] = stringTokenizer.nextToken();
    boolean invalidFormat = false;

    // Empty xpointer component is not supported
    if(xpointerPath[i].length() == 0){
      invalidFormat = true;
    }

    if(i > 0){
      try {
        Integer.parseInt(xpointerPath[i]);
      } catch (NumberFormatException e) {
        invalidFormat = true;
      }
    }

    if(invalidFormat){
      throw new ElementLocatorException(
        "Only the element() scheme is supported when locating XPointer links."
        + "Supported formats: element(elementID), element(/1/2/3),
              element(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

uri	The namespace URI, or the empty string if the element has no namespace URI or if namespace processing is disabled.
localName	Local name of the element.
qName	Qualified name of the element.
atts	Attributes attached to the element. If there are no attributes, this argument will be empty.

The method returns `true` if the processed element is found to be the one indicated by the link.

The `XPointerElementLocator` implementation of the `startElement` will update the depth of the current element and keep the index of the element in its parent. If the `xpointerPath` starts with an element ID then the

current element ID is verified to match the specified ID. If this is the case the depth of the XPointer is updated taking into account the depth of the current element.

If the XPointer path depth is the same as the current element depth then the kept indices of the current element path are compared to the indices in the XPointer path. If all of them match then the element has been found.

```
public boolean startElement(String uri, String localName,
        String name, Attr[] 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 `DefaultElementLocator`, `IDElementLocator` or `XPointerElementLocator` can be found in the Oxygen Default Frameworks project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen website*.

Configuring a custom Drag and Drop listener

You can add your own drag and drop listener implementation of `ro.sync.ecss.extensions.api.DnDHandler`. You can choose from three interfaces to implement depending on whether you are using the framework with the Oxygen Eclipse plugin or the standalone version or if you want to add the handler for the Text or Author pages.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen website*. Also it can be downloaded as a *zip archive from the website*.

Table 2: Interfaces for the DnD listener

Interface	Description
`ro.sync.exml.editor.xmleditor.pageauthor.AuthorCustomDnDHandler`	Receives callbacks from the Oxygen standalone application for Drag And Drop in Author mode.
`com.oxygenxml.editor.editors.author.AuthorDnDListener`	Receives callbacks from the Oxygen Eclipse plugin for Drag And Drop in Author mode.
`com.oxygenxml.editor.editors.TextDnDListener`	Receives callbacks from the Oxygen 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 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 {
```

2. The `hasReferences` method verifies if the handler considers the node to have references. It takes as argument an `AuthorNode` that represents the node which will be verified. The method will return `true` if the node is considered to have references. In our case, to be a reference the node must be an element with the name *ref* and it must have an attribute named *location*.

```
public boolean hasReferences(AuthorNode node) {
  boolean hasReferences = false;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      hasReferences = attrValue != null;
    }
  }
  return hasReferences;
}
```

3. The method `getDisplayName` returns the display name of the node that contains the expanded 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;
}
```

4. 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 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;
}
```

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 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:

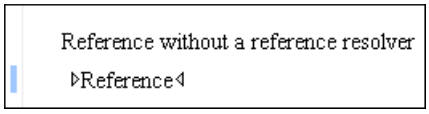


Figure 198: Reference with no specified reference resolver

When the above implementation is configured, the reference has the expected layout:

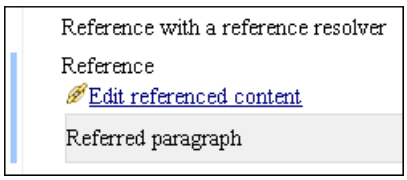


Figure 199: Reference with reference resolver

Configuring CSS Styles Filter

You can modify the CSS styles for each `ro.sync.ecss.extensions.api.node.AuthorNode` rendered in the Author page 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 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 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 a Table Column Width Provider

In the documentation framework the `table` element as well as the table columns can have specified widths. In order for these widths to be considered by Oxygen Author we need to provide the means to determine them. As explained in the *Styling the Table Element* section which describes the CSS properties needed for defining a table, if you use the table element attribute **width** Oxygen 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 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 `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 `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 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 row_span="2">cs=1, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

When no table column width provider is specified, the table has the following layout:

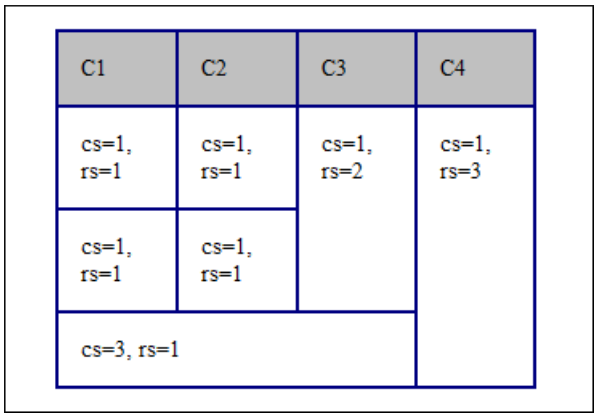


Figure 200: Table layout when no column width provider is specified

When the above implementation is configured, the table has the correct layout:

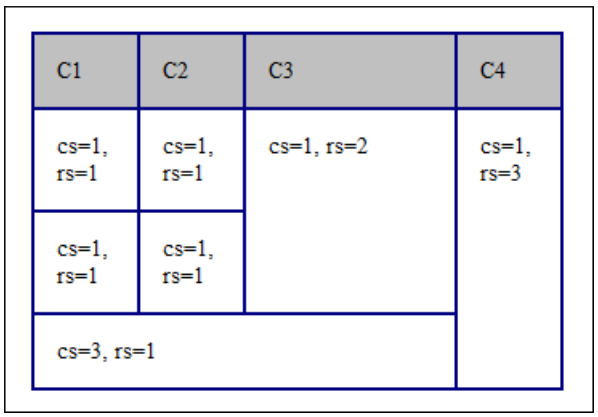


Figure 201: Columns with custom widths

Configuring a Table Cell Span Provider

In the documentation framework the `table` element can have cells that span over multiple columns and rows. As explained in the *Styling the Table Element* section which describes the CSS properties needed for defining a table, you need to indicate Oxygen Author a method to determine the cell spanning. If you use the cell element attributes **rowspan** and **colspan** or **rows** and **cols**, Oxygen 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 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 `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 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:

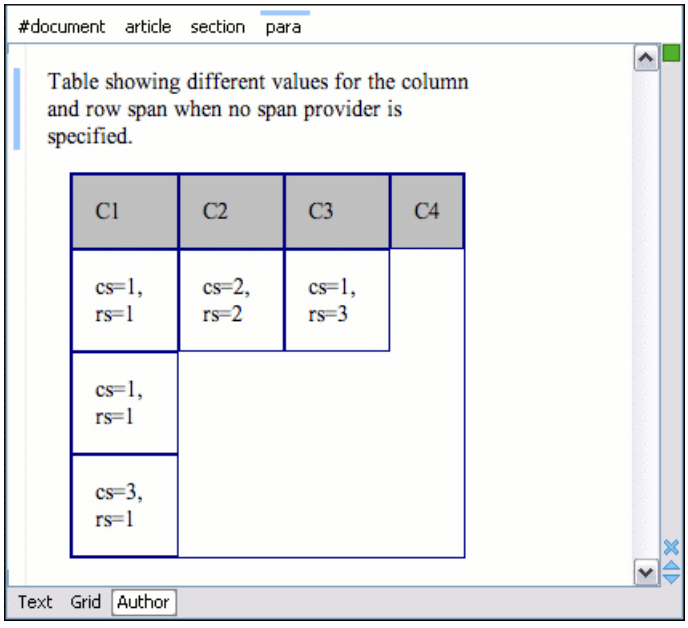


Figure 202: Table layout when no cell span provider is specified

When the above implementation is configured, the table has the correct layout:

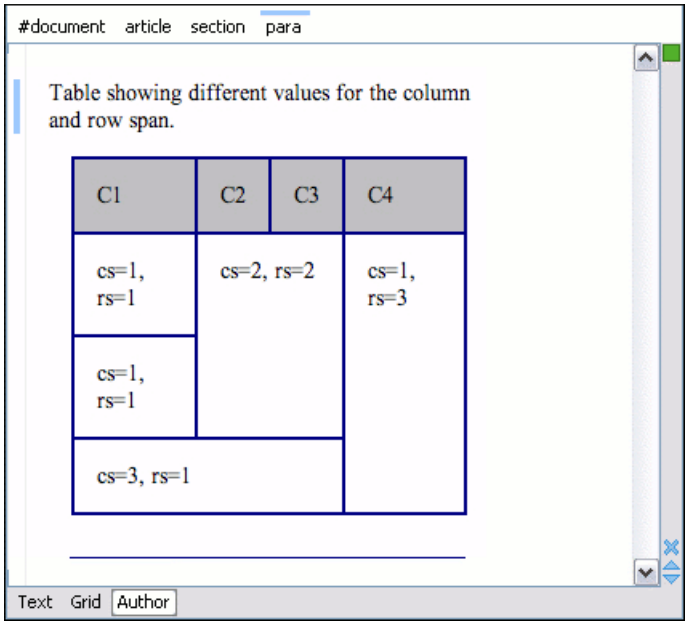


Figure 203: Cells spanning multiple rows and columns.

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 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:

```
/**
 * Assign unique IDs between a start
 * and an end offset in the document
 * @param startOffset Start offset
 * @param endOffset End offset
 */
void assignUniqueIDs(int startOffset, int endOffset);

/**
 * @return true if auto
 */
boolean 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:

```
  /**
   * Check if the attribute specified by QName can
   * be considered as a valid attribute to copy
   * when the element is split.
   *
```

```
   * @param attrQName The attribute qualified name
   * @param element The element
   * @return true if the attribute should be copied
   * when Split is performed.
   */
  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.

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. 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. The user role must be set to *Developer* otherwise a warning is displayed and a duplicate copy of the DITA document type is created and edited. This check makes sure that regular content authors who just edit the content of XML documents do not accidentally modify the document type. 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`.

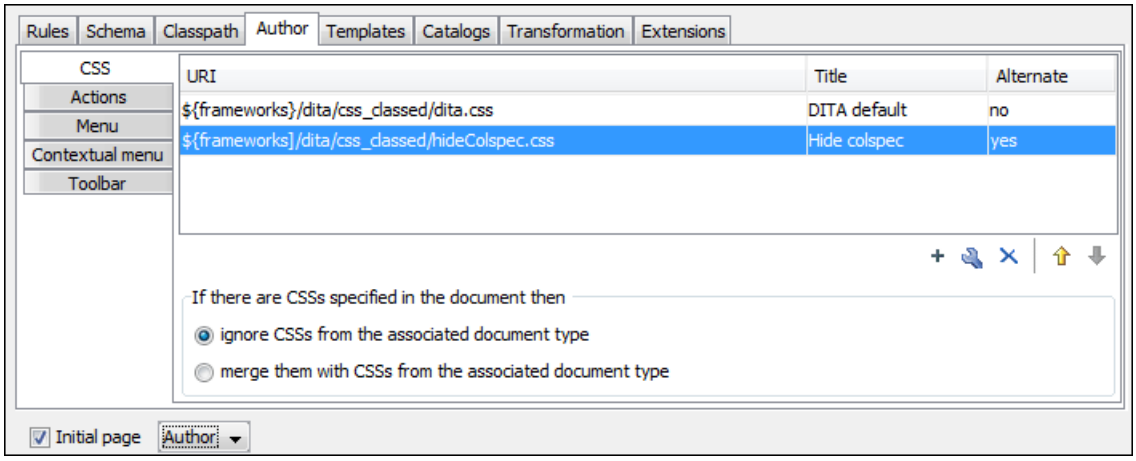


Figure 204: Set the location of the default CSS stylesheet

3. 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 has support for allowing you to share the customizations which you have made 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 in two ways:

- Save it externally in a separate framework folder in the `OXYGEN_INSTALL_DIR/frameworks` directory.

  👉 **Important:** In order for this approach to work you will need to have Oxygen installed to a folder with full write access.

  Please see the following steps:

  1. Create a new directory in the `OXYGEN_INSTALL_DIR/frameworks` for your new 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. Switch the user role to **Developer** in the Oxygen Preferences *Document Type Association* page.
  3. Create your new custom document type and save it external in the newly created framework directory (with a name like `custom.framework`).
  4. 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.
  5. If everything went fine then you should have a new configuration file saved in: `OXYGEN_INSTALL_DIR/frameworks/your_framework_dir/custom.framework` after the Preferences are saved.
  6. You can then share the new framework directory with other users (have them copy it to their `OXYGEN_INSTALL_DIR/frameworks` directory) and the new document type will be available in the list of Document Types when Oxygen is started.

- Save the document type at project level in the *Document Type Association* page.

  Please see the following steps:

  1. Create a new directory with full write access somewhere on your local drive which will contain the Oxygen project file and associated document type resources (CSS files, new file templates, schemas used for validation, catalogs).
  2. From the Oxygen *Project view* create a new project and save it in the newly created directory.
  3. In the Oxygen Preferences *Document Type Association* page switch the radio button at the bottom of the page to **Project Options**.
  4. Create your new custom document type using the default **internal** storage for it. It will actually be saved in the previously chosen Oxygen project `.xpr` file.
  5. Configure the custom document type according to your needs, take special care to make all file references relative to the project directory by using the `${pd}` editor variable. The *Author Developer Guide* contains all details necessary for creating and configuring a new document type.
  6. You can then share the new project directory with other users and if they open in the *Project view* the customized project then the new document type will be available in the list of Document Types.

CSS Support in Oxygen 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.

CSS 2.1 Features

This section enumerates the CSS 2.1 features that are supported by Oxygen XML Author.

Supported CSS 2.1 Selectors

Expression	Name	Description / Example
`*`	Universal selector	Matches any element
`E`	Type selector	Matches any `E` element (i. e. an element with the local name `E`)
`E F`	Descendant selector	Matches any `F` element that is a descendant of an `E` element.
`E > F`	Child selectors	Matches any `F` element that is a child of an element `E`.
`E:first-child`	The `:first-child` pseudo-class	Matches element `E` when `E` is the first child of its parent.
`E:lang(c)`	The `:lang()` pseudo-class	Matches element of type `E` if it is in (human) language `c` (the document language specifies how language is determined).
`E + F`	Adjacent selector	Matches any `F` element immediately preceded by a sibling element `E`.
`E[foo]`	Attribute selector	Matches any `E` element with the `"foo"` attribute set (whatever the value).
`E[foo="warning"]`	Attribute selector	Matches any `E` element whose `"foo"` attribute value is exactly equal to `"warning"`.
`E[foo~="warning"]`	Attribute selector	Matches any `E` element whose `"foo"` attribute value is a list of space-separated values, one of which is exactly equal to `"warning"`.
`E[lang|="en"]`	Attribute selector	Matches any `E` element whose `"lang"` attribute has a hyphen-separated list of values beginning (from the left) with `"en"`.
`E:before` and `E:after`	Pseudo elements	The `':before'` and `':after'` pseudo-elements can be used to insert generated content before or after an element's content.

Unsupported CSS 2.1 Selectors

Expression	Name	Description / Example
E#myid	ID selectors	Matches any E element with ID equal to "myid".

Expression	Name	Description / Example
E:link, E:visited	The link pseudo-class	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited).
E:active, E:hover, E:focus	The dynamic pseudo-classes	Matches E during certain user actions.
E:first-line	The :first-line pseudo-class	The :first-line pseudo-element applies special styles to the contents of the first formatted line of a paragraph.
E:first-letter	The :first-letter pseudo-class	The :first-letter pseudo-element must select the first letter of the first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line. The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects.

CSS 2.1 Properties

All the properties belonging to the *aural* and *paged* categories are **not supported** in Oxygen Author. The properties from the table below belong to the *visual* category.

Name	Supported Values	Not Supported Values
`'background-attachment'`		ALL
`'background-color'`	`<color> | inherit`	`transparent`
`'background-image'`		ALL
`'background-position'`		ALL
`'background-repeat'`		ALL
`'background'`		ALL
`'border-collapse'`		ALL
`'border-color'`	`<color> | inherit`	`transparent`
`'border-spacing'`		ALL
`'border-style'`	`<border-style> | inherit`	
`'border-top' 'border-right' 'border-bottom' 'border-left'`	`[ <border-width> || <border-style> || 'border-top-color' ] | inherit`	
`'border-top-color' 'border-right-color' 'border-bottom-color' 'border-left-color'`	`<color> | inherit`	`transparent`
`'border-top-style' 'border-right-style' 'border-bottom-style' 'border-left-style'`	`<border-style> | inherit`	
`'border-top-width' 'border-right-width' 'border-bottom-width' 'border-left-width'`	`<border-width> | inherit`	

Name	Supported Values	Not Supported Values
'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
'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

Name	Supported Values	Not Supported Values
'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 | 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	
'margin'	<margin-width> | inherit | auto	
'max-height'		ALL
'max-width'	<length> | <percentage> | none | inherit - supported for block-level and replaced elements, e.g. images, tables, table cells.	
'min-height'		ALL
'min-width'	<length> | <percentage> | inherit - supported for block-level and replaced elements, e. g. images, tables, table cells.	
'outline-color'		ALL
'outline-style'		ALL
'outline-width'		ALL

Name	Supported Values	Not Supported Values
'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'		ALL
'vertical-align'	baseline | sub | super | top | text-top | middle | bottom | text-bottom | inherit	<percentage> | <length>
'visibility'	visible | hidden | inherit	collapse
'white-space'	normal | pre | nowrap | pre-wrap | pre-line	
'width'	<length> | <percentage> | auto | inherit - supported for block-level and replaced elements, e.g. images, tables, table cells.	
'word-spacing'		ALL
'z-index'		ALL

CSS 3 Features

This section enumerates the CSS 3 features that are supported by Oxygen XML Author.

CSS 3 Namespace Selectors

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 Author uses a different approach similar to the CSS Level 3 specification. If the element name from the CSS selector is not preceded by a namespace prefix it is considered to match an element with the same local name as the selector value and ANY namespace, otherwise the element must match both the local name and the namespace.

In CSS up to version 2.1 the name tokens from selectors are matching all elements from ANY namespace that have the same local name. Example:

```
<x:b xmlns:x="ns_x"/>
<y:b xmlns:y="ns_y"/>
```

Are both matched by the rule:

```
b {font-weight:bold}
```

Starting with CSS Level 3 you can create selectors that are namespace aware.

Defining both prefixed namespaces and the default namespace

Given the namespace declarations:

```
@namespace sync "http://sync.example.org";
@namespace "http://example.com/foo";
```

In a context where the default namespace applies:

sync|A	represents the name A in the `http://sync.example.org` namespace.
|B	represents the name B that belongs to NO NAMESPACE.
***|C**	represents the name C in ANY namespace, including NO NAMESPACE.
D	represents the name D in the `http://example.com/foo` namespace.

Defining only prefixed namespaces

Given the namespace declaration:

```
@namespace sync "http://sync.example.org";
```

Then:

sync|A	represents the name A in the `http://sync.example.org` namespace.
|B	represents the name B that belongs to NO NAMESPACE.
***|C**	represents the name C in ANY namespace, including NO NAMESPACE.
D	represents the name D in ANY namespace, including NO NAMESPACE.

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 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 Author accepts one of the following types:

`color` The value represents a color. The attribute may specify a color in different formats. Oxygen 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 Author supports only images. The attribute value can be a complete URL, or a relative one to the XML document. Please note that this URL is also resolved through the catalog resolver.

`integer` The value must be interpreted as an integer.

`number` The value must be interpreted as a float number.

`length` The value must be interpreted as an integer.

`percentage` The value must be interpreted relative to another value (length, size) expressed in percents.

`em` The value must be interpreted as a size. 1 em is equal to the *font-size* of the relevant font.

`ex` The value must be interpreted as a size. 1 ex is equal to the *height* of the **x** character of the relevant font.

`px` The value must be interpreted as a size expressed in pixels relative to the viewing device.

`mm` The value must be interpreted as a size expressed in millimeters.

`cm` The value must be interpreted as a size expressed in centimeters.

`in` The value must be interpreted as a size expressed in inches. 1 inch is equal to 2.54 centimeters.

`pt` The value must be interpreted as a size expressed in points. The points used by CSS2 are equal to 1/72th of an inch.

pc The value must be interpreted as a size expressed in picas. 1 pica is equal to 12 points.

`default_value` This argument specifies a value that is used by default if the attribute value is missing. This argument is optional.

Usage samples for the attr() function

Consider the following XML instance:

```
<sample>
    <para bg_color="#AAAAFF">Blue paragraph.</para>
    <para bg_color="red">Red paragraph.</para>
    <para bg_color="red" font_size="2">Red paragraph with large
font.</para>
    <para bg_color="#00AA00" font_size="0.8" space="4">
        Green paragraph with small font and margin.</para>
</sample>
```

The `para` elements have `bg_color` attributes with RGB color values like #AAAAFF. You can use the `attr()` function to change the elements appearance in the editor based on the value of this attribute:

```
background-color:attr(bg_color, color);
```

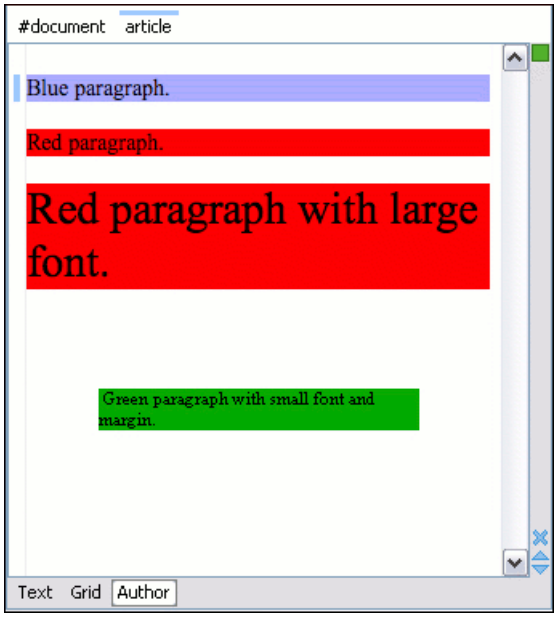
The attribute `font_size` represents the font size in *em* units. You can use this value to change the style of the element:

```
font-size:attr(font_size, em);
```

The complete CSS rule is:

```
para{
 display:block;
 background-color:attr(bg_color, color);
 font-size:attr(font_size, em);
 margin:attr(space, em);
}
```

The document is rendered as:



Additional Custom Selectors

Oxygen Author provides support for selecting additional types of nodes. These custom selectors apply to: *document*, *doctype sections*, *processing-instructions*, *comments*, *CDATA sections*, and *entities*. In order for the custom selectors to work in your CSSs you will have to declare the Author extensions namespace at the beginning of the stylesheet documents:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
```

Example rules:

- *document:*

```
oxy|document {
    display:block;
}
```

- *doctype sections:*

```
oxy|doctype {
    display:block;
    color:blue;
    background-color:transparent;
}
```

- *processing-instructions:*

```
oxy|processing-instruction {
    display:block;
    color:purple;
    background-color:transparent;
}
```

- *comments:*

```
oxy|comment {
    display:block;
    color:green;
    background-color:transparent;
}
```

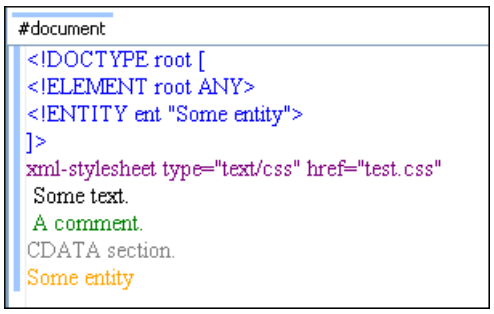
- *CDATA sections:*

```
oxy|cdata{
    display:block;
    color:gray;
    background-color:transparent;
}
```

- *entities:*

```
oxy|entity {
    display:morph;
    editable:false;
    color:orange;
    background-color:transparent;
}
```

A sample document rendered using these rules:



Oxygen CSS Extensions

CSS stylesheets provide support mainly for displaying documents. When editing documents some extensions of the W3C CSS specification 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 a URL from a relative path location
- string processing functions

Media Type oxygen

The style sheets can specify how a document is to be presented on different media: on the screen, on paper, speech synthesizer, etc. You can specify that some of the features of your CSS stylesheet should be taken into account only in the Oxygen Author and ignored in the rest. This can be accomplished by using the media type oxygen.

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 who does not recognize `@media oxygen` and bold and underlined in Oxygen Author.

You can use this media type to group specific Oxygen CSS features and also to hide them when opening the documents with other viewers.

Folding Elements: `foldable` and `not-foldable-child` Properties

Oxygen 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.

To define the element whose content can be folded by the user, you must use the property: `foldable:true;`.

When collapsing an element, it is useful to keep some of its content visible, like a short description of the collapsed region. The property `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 (`foldable:true;`) but it doesn't have the property `not-foldable-child` or none of the specified non-foldable children exists then the element will still be foldable. In this case the element that will be kept visible when folded will be the **before** pseudo-element.

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 {
    foldable:true;
    not-foldable-child: title;
}
```

Link Elements

Oxygen 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{
    link: attr(targetdoc);
    content: "Click to open: " attr(targetdoc);
}
```

Display Tag Markers

Oxygen 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 `display-tags`. Its possible values are :

- *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.

```
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{
    display-tags: none;
    display:block;
    margin: 0.5em 0;
}
```

Oxygen Custom CSS Functions

In Oxygen Author there are implemented a few Oxygen specific custom CSS functions. Imbricated custom functions are also supported.

Imbricated functions

The result of the functions below will be the local name of the current node with the first letter capitalized.

```
capitalize(local-name())
```

The `local-name()` Function

This function evaluates the local name of the current node. It does not have any arguments.

The `name()` Function

This function evaluates the qualified name of the current node. It does not have any arguments.

The `url()` function

This function evaluates the URL of a location relative to the CSS file location and appends each of the relative path components to the final location.

```
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 `base-uri()` Function

This 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.

The `parent-url()` Function

This function evaluates the parent URL of an URL received as string.

```
parent-url ( URL )
```

URL	The URL as string.

The `capitalize()` Function

This function capitalizes the first letter of the text received as argument.

```
capitalize ( text )
```

text	The text for which the first letter will be capitalized.

The `uppercase()` Function

This function transforms to upper case the text received as argument.

```
uppercase ( text )
```

text	The text to be capitalized.

The `lowercase()` Function

This function transforms to lower case the text received as argument.

```
lowercase ( text )
```

text	The text to be lower cased.

The `concat()` Function

This function concatenates the received string arguments.

```
concat ( str_1 , str_2 )
```

str_1 ... str_n	The string arguments to be concatenated.

The `replace()` Function

This function has two signatures:

- `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.

- `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.

The `unparsed-entity-uri()` Function

This function returns the URI value of an unparsed entity name.

`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: url(unparsed-entity-uri(attr(entityref)));
}
```

The `attributes()` Function

This function concatenates the attributes for an element and returns the serialization.

`attributes ( )`

attributes()

For the following XML fragment:`<element att1="x" xmlns:a="2" x="""/>` the `attributes()` function will return `att1="x" xmlns:a="2" x="""`.

The `substring()` Function

This function has two signatures:

- `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.

`substring('abcd', 1)` returns the string `'bcd'`.

`substring('abcd', 4)` returns an empty string.

`substring('abcd', 1, 3)` returns the string `'bc'`.

The `indexof()` Function

This function has two signatures:

- `indexof ( text , toFind )`

  Returns the index within **text** string of the first occurrence of the **toFind** substring.

text	Text to search in.
toFind	The searched substring.

- `indexof( text , toFind , fromOffset )`

  Returns the index within **text** string of the first occurrence of the **toFind** substring. The search starts from **fromOffset** index.

text	Text to search in.
toFind	The searched substring.
fromOffset	The index from which to start the search.

  `indexof('abcd', 'bc')` returns 1.

  `indexof('abcdbc', 'bc', 2)` returns 4.

The `lastindexof()` Function

This function has two signatures:

- `lastindexof( text , toFind )`

  Returns the index within **text** string of the rightmost occurrence of the **toFind** substring.

text	Text to search in.
toFind	The searched substring.

- `lastindexof( text , toFind , fromOffset )`

  The search starts from **fromOffset** index. Returns the index within **text** string of the last occurrence of the **toFind** substring, searching backwards starting from the **fromOffset** index.

text	Text to search in.
toFind	The searched substring.
fromOffset	The index from which to start the search backwards.

  `lastindexof('abcdbc', 'bc')` returns 4.

  `lastindexof('abcdbccdbc', 'bc', 2)` returns 1.

The `xpath()` Function

This function has one signature:

- `xpath( expression )`

  Evaluates the given XPath expression and returns the result.

expression	XPath expression to be evaluated.

  The following example counts the number of words from a paragraph and displays the result in front of it:

```
para:before{ content: concat("|Number of words:",
xpath("count(tokenize(normalize-space(string-join(text(), '')), '
'))"), "| "); }
```

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{
    foldable:true;
    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>
                <header>
                    <td>XSLT Elements</td>
                    <td>Description</td>
                </header>
                <tr>
                    <td>
                        <b>xsl:stylesheet</b>
                    </td>
                    <td>The <i>xsl:stylesheet</i> element is
                        always the top-level element of an
                        XSL stylesheet. The name
                            <i>xsl:transform</i> may be used
                        as a synonym.</td>
                </tr>
                <tr>
                    <td>
                        <b>xsl:template</b>
                    </td>
                    <td>The <i>xsl:template</i> element has
                        an optional mode attribute. If this
                        is present, the template will only
                        be matched when the same mode is
                        used in the invoking
                            <i>xsl:apply-templates</i>
                        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 column_span="2">End of the list</td>
                </tr>
            </table>
        </section>
        <section>
            <title>XPath</title>
            <abs:def>XPath (XML Path Language) is a terse
                (non-XML) syntax for addressing portions of
                an XML document. </abs:def>
            <para>Some of the XPath functions.</para>
            <table>
                <header>
                    <td>Function</td>
                    <td>Description</td>
                </header>
                <tr>
                    <td>format-number</td>
                    <td>The <i>format-number</i> function
                        converts its first argument to a
                        string using the format pattern
                        string specified by the second
                        argument and the decimal-format
                        named by the third argument, or the
                        default decimal-format, if there is
                        no third argument</td>
                </tr>
                <tr>
                    <td>current</td>
                    <td>The <i>current</i> function returns
                        a node-set that has the current node
                        as its only member.</td>
                </tr>
                <tr>
                    <td>generate-id</td>
                    <td>The <i>generate-id</i> function
                        returns a string that uniquely
                        identifies the node in the argument
                        node-set that is first in document
                        order.</td>
                </tr>
            </table>
        </section>
    </section>
    <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">
        <img src="{@href}"/>
    </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>
```

Oxygen XML Author Component

The Oxygen XML Author component was designed as a separate product to provide the functionality of the standard Author page. 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 SDK for Java/Swing integrations is available online on the oXygen XML website: *http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-sample.zip*

Licensing

Just like the oXygen standard deployment, the Author component requires license information in order to run. Licensing information must follow the same models imposed for the standard oXygen application, namely the floating, named-user based or group licenses. You can license an Author component using standard oXygen XML Editor/Author license keys.

You can set the component to:

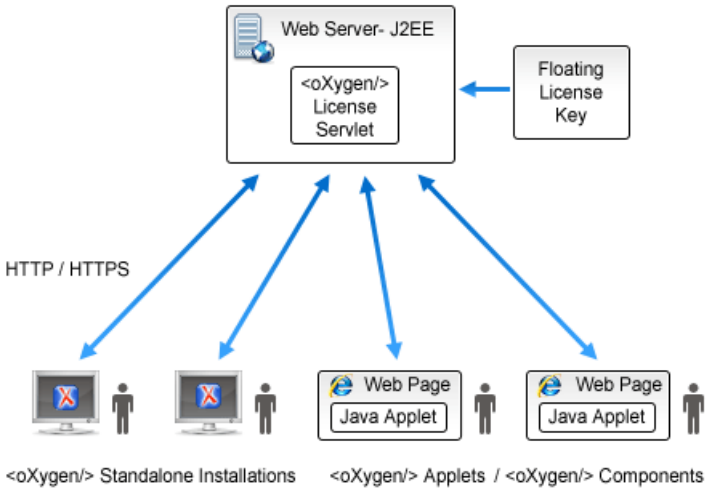
- display the license registration dialog to the end user. This is the default behavior and transfers the licensing responsibility to the end-user. The standard licensing procedure applies.
- programmatically inject the licensing information directly in the component. This is especially useful when you use a multiple-user or group license.

  : You must make sure only the authorized users will access the application.

- programmatically set-up floating license server details.

The most common use-case is when you as a developer customize the component and then want to deliver it to end users (either embedded in a Java application or a Java Web applet). Your licensing options are:

- named-user based model, where users provide their own oXygen license keys and register the component;
- floating license model, where the component comes pre-configured to use one of the oXygen floating license servers (either the standalone or the servlet version).



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);
- The applet was tested for compatibility with the following browsers:

	IE 7	IE 8 (32bit)	IE 9 (64bit)	Firefox 3	Firefox 4	Safari 5	Chrome
Windows XP	Passed	Passed	-	Passed	Passed	-	Passed
Vista	Failed	Passed	Failed	Passed	Passed	Failed	Passed
Windows 7	-	-	Passed	Passed	Passed	-	Passed
Mac OS X 10.6	-	-	-	Failed	Passed	Passed	Failed
Linux Ubuntu 10	-	-	-	Failed	-	-	Failed

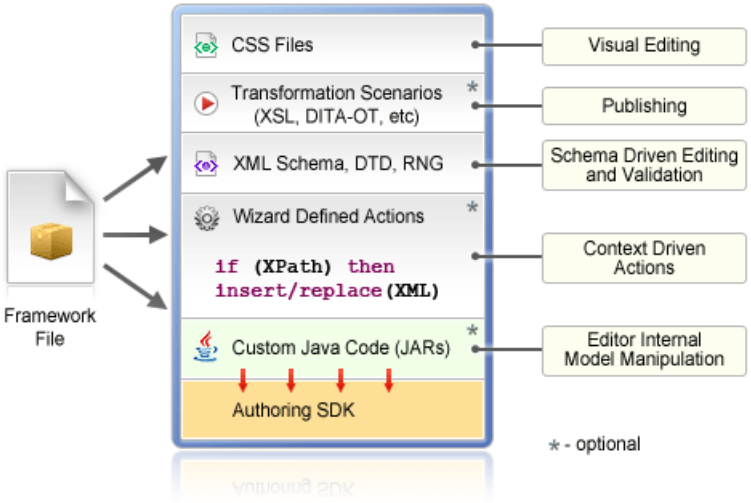
Running the Author component embedded in a third-party Java/Swing application requires:

- Oracle (Sun) Java JRE version 1.6 or newer;
- At least 100 MB disk space and 100MB free memory;

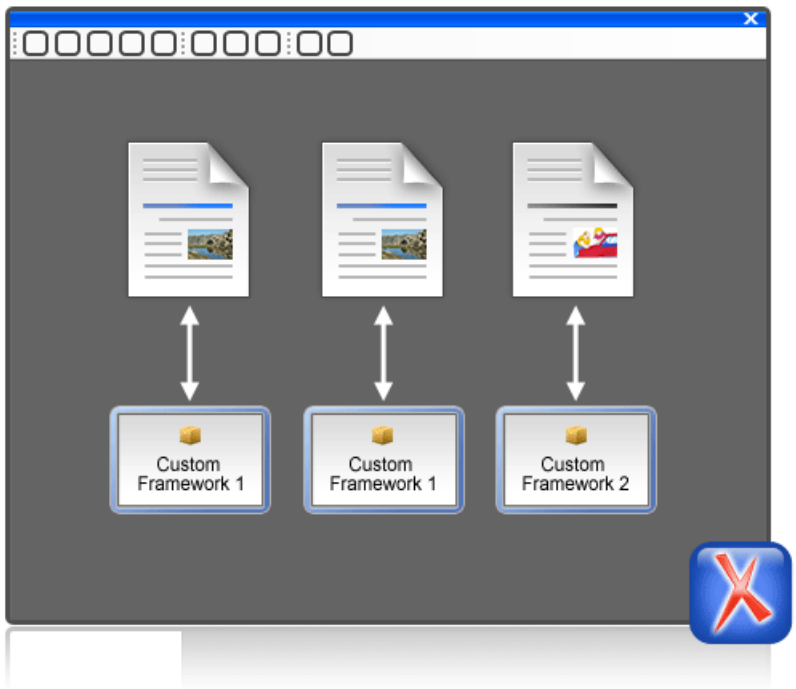
Customization

For a special type of XML, you can create a custom framework (which also works in an Oxygen standalone version). Oxygen 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 application for editing in the Author page. 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.

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.

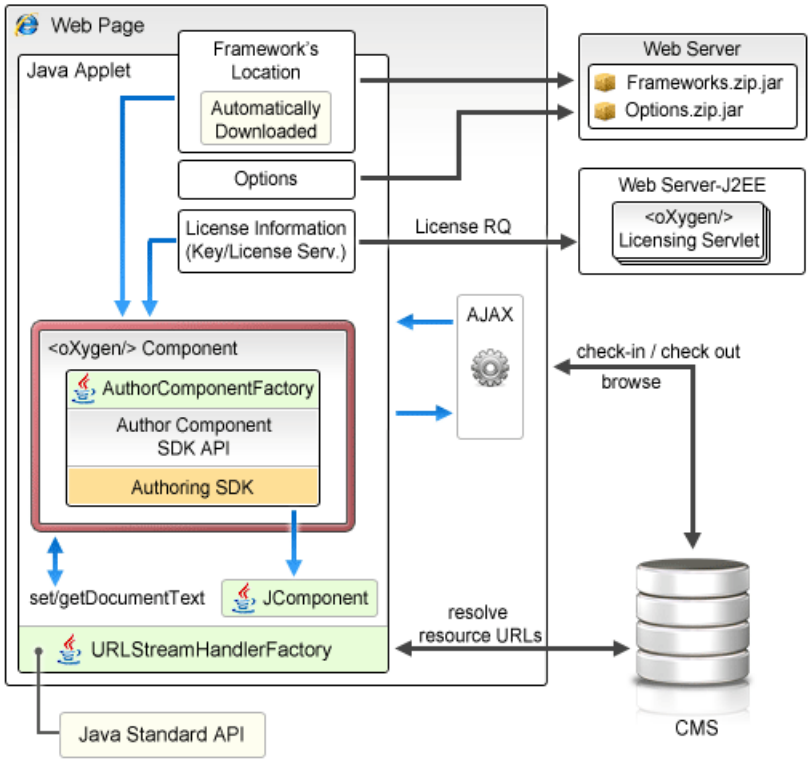


Figure 205: Oxygen Author Component deployed as a Java applet

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.
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.

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 check 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`.

Chapter 9

Grid Editor

Topics:

- *Layouts: Grid and Tree*
- *Navigating the Grid*
- *Specific Grid Actions*
- *Drag and Drop in the Grid Editor*
- *Copy and Paste in the Grid Editor*
- *Bidirectional Text Support in the Grid Editor*

In the grid editor the XML document is displayed as a structured grid of nested tables in which the text content can be modified by non technical users without editing directly the XML tags. The tables can be expanded and collapsed with a mouse click to show or hide the elements of the document as needed. Also the document structure can be changed easily with drag and drop operations on the grid components. The tables can be zoomed using **(Ctrl - +)** , **(Ctrl - -)**, **(Ctrl - 0)** or **(Ctrl - mouse wheel)**.

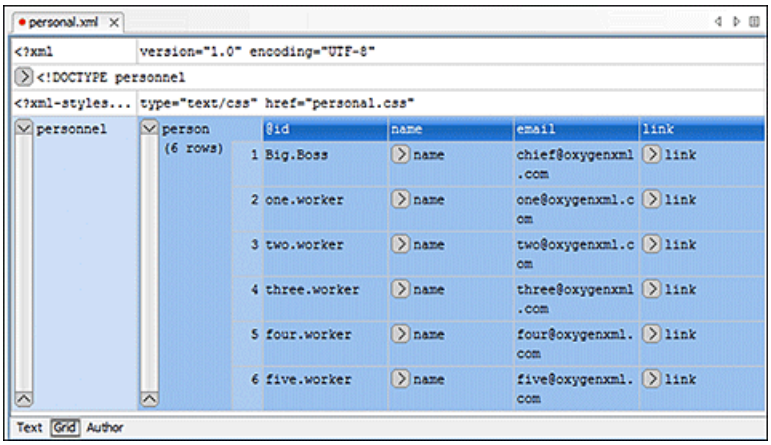


Figure 206: The Grid Editor

You can switch between the text tab and the grid tab of the editor panel with the two buttons **Text** and **Grid** available at the bottom of the editor panel. Also the switch can be performed with the actions **Document** > **Edit mode** > **Grid** and **Document** > **Edit mode** > **Text** .

If the edited document is associated with a schema (DTD, XML Schema, Relax NG, etc.), the editor offers content completion for the element and attributes names and values. If you choose to insert an element that has required content, it will be inserted automatically including the subtree of needed elements and attributes.

To display the content completion popup you have to start editing, for example by double clicking the cell. When editing, pressing **(Ctrl - Space)** redisplays the popup.

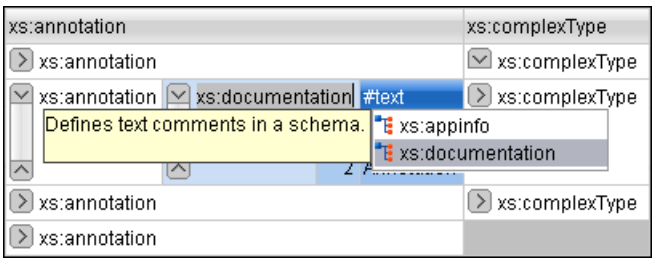


Figure 207: Content Completion in Grid Editor

Layouts: Grid and Tree

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

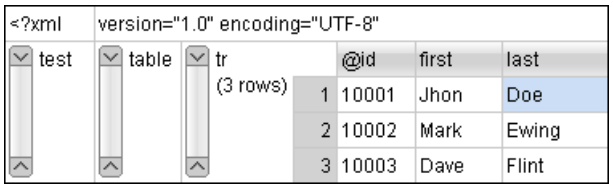


Figure 208: Grid Layout

The other layout mode is tree-like. This layout does not create any table. It presents the structure of the document directly.

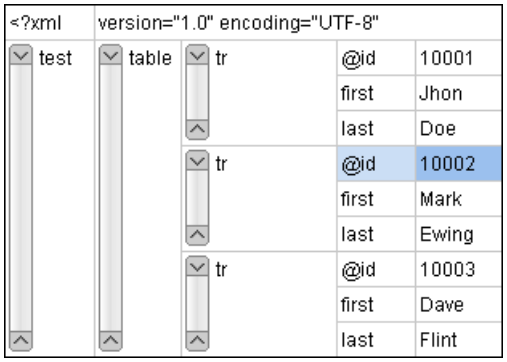


Figure 209: Tree Layout

You can switch between the two modes using the menu **Document** > **Grid Layout** > **Grid mode/Tree mode** .

Navigating the Grid

When you open a document first in the grid tab, the document is collapsed so that it shows just the root element and its attributes. The grid disposition of the node names and values are very similar to a web form or a dialog. The same set of key shortcuts used to select dialog components are used in the grid. For instance moving to the next editable value in a table row is done using the **(Tab)** key. Moving to the previous cell employs the **(Shift-Tab)** key. Changing a value assumes pressing the **(Enter)** key or start typing directly the new value, and, when the editing is finished, pressing **(Enter)** again to commit the data into the document.

The arrow keys and the **(Page Up/Down)** keys can be used for navigation. By pressing **(Shift)** while using these keys you can create a selection zone. To add other nodes that are not close to this zone, you can use the mouse and the **(Ctrl)** key (**(Command)** on Mac OS X).

The following key combination may be used to scroll the grid:

- Ctrl - Up scrolls the grid upwards
- Ctrl - Down scrolls the grid downwards
- Ctrl - Left scrolls the grid to the left
- Ctrl - Right scrolls the grid to the right

A left arrow sign displayed to the left of the node name indicates that this node has child nodes. You can click this sign to display the children. The expand/collapse actions can be also invoked by pressing the **(NumPad Plus)** and **(NumPad Minus)** keys.

A set of expand/collapse actions can be accessed from the submenu **Expand/Collapse** of the contextual menu.

The same actions can be accessed from the menu **Document** > **Grid Expand/Collapse** .

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

- **Expand All** - Expands the selection and all its children.
- **Collapse All** - Collapses the selection and all its children.
- **Expand Children** - Expands all the children of the selection but not the selection.
- **Collapse Children** - Collapses all the children of the selection but not the selection.
- **Collapse Others** - Collapses all the siblings of the current selection but not the selection.

Specific Grid Actions

In order to access these actions you can click the column header and choose from the contextual menu the item **Table** . The same set of actions are available in the menu **Document** 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:

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. If there is present a mixed set of values in the column, a dialog will be displayed allowing to choose the desired type between numerical and alphabetical.

Inserting a Row in a Table

You can add a row by either a copy/paste operation over a row, or directly, by invoking the action from the contextual menu: **Table** > **Insert row** .

A shorter way of inserting a new row is to move the selection over the row header, and then to press **(Enter)**. The row header is the zone in the left of the row that holds the row number. The inserted row will be below the selection.

Inserting a Column in a Table

You can insert a column after the selected one, using the action from the contextual menu **Table** > **Insert column** .

Clearing the Content of a Column

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

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 contextual menu: **Duplicate** and in the menu **Document** > **Grid Edit** > **Duplicate** .

Refresh Layout

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

Start Editing a Cell Value

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

The action is found in the menu **Document** > **Grid Edit** > **Start Editing** .

Stop Editing a Cell Value

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

The action is found in the menu **Document** > **Grid Edit** > **End Editing** .

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

Drag and Drop in the Grid Editor

The drag and drop features of the grid editor make easy the arrangement of the different sections in your XML document.

Using 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 single selection and multiple selection.

Note that when dragging the editor paints guide-lines showing accepted locations where the nodes can be dropped.

Nodes can be dragged outside the grid editor and text from other applications can be dropped inside the grid. See *Copy and Paste in the Grid Editor* for details.

Copy and Paste in the Grid Editor

The selection in the grid is a bit complex relative 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 the user using the mouse, or are 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; the editor automatically extends the selection so it contains also 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. Pasting these nodes may be done in two ways, relative to the current selected cell: by default as brother, just below (after) , or as last child of the selected cell.

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

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

The copied nodes from the grid can be pasted also into the text editor or other applications. When copying from 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.

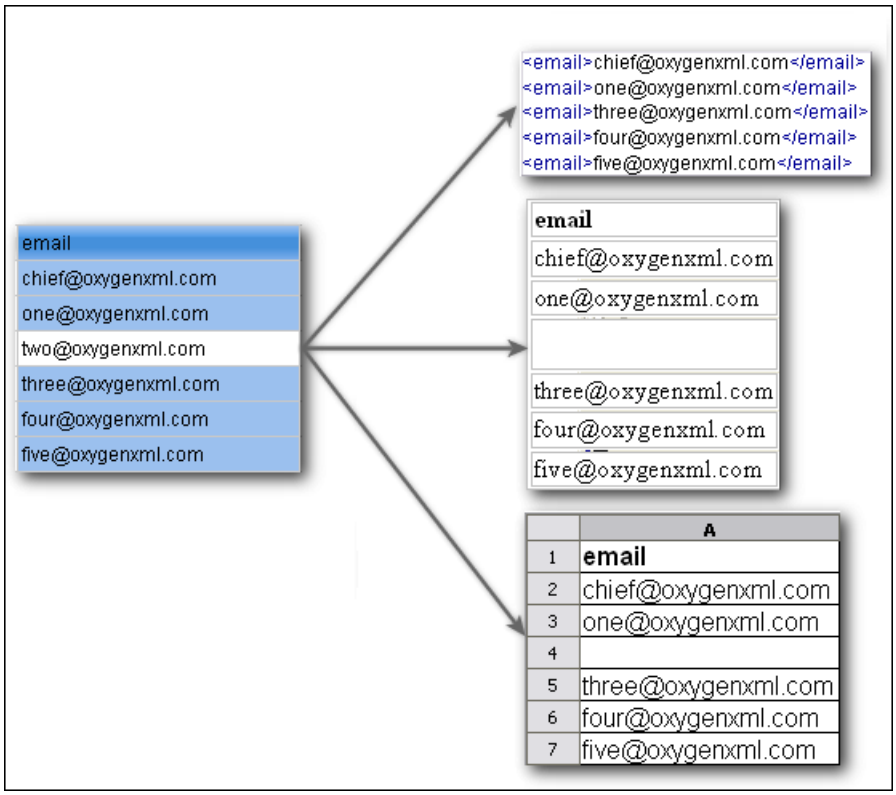


Figure 210: Copying from grid to other editors

In the grid editor you can paste wellformed 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.

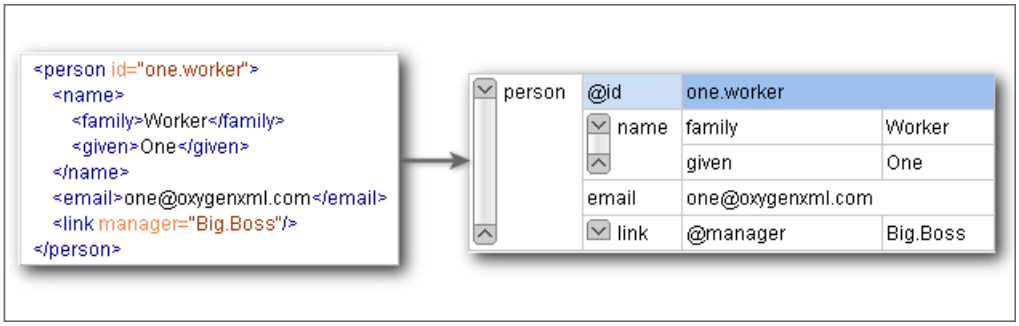


Figure 211: Copying XML data into grid

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 values from these cells will be overwritten and new ones will be created if needed. This is useful for example when trying to transfer data from Excel like editors into grid editor.

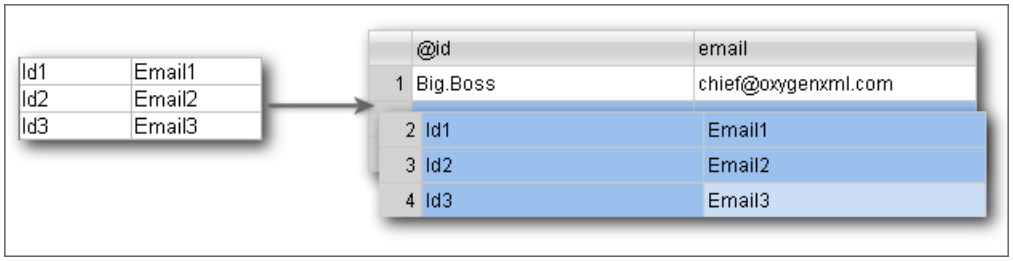


Figure 212: Copying tab separated values into grid

Bidirectional Text Support in the Grid Editor

If you are editing documents employing a different text orientation you can change the way text is rendered and edited in the grid cells. For this, you can use the shortcut **(Ctrl-Shift-O)** to toggle from the default left to right text orientation to the right to left orientation.

Note that this change applies only to the text from the cells, not to the layout of the grid editor.

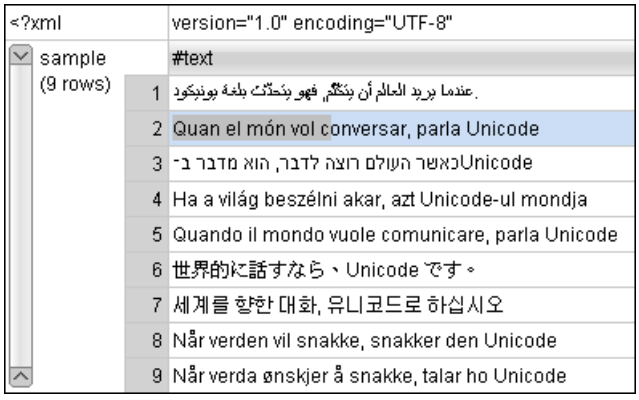


Figure 213: Default left to right text orientation

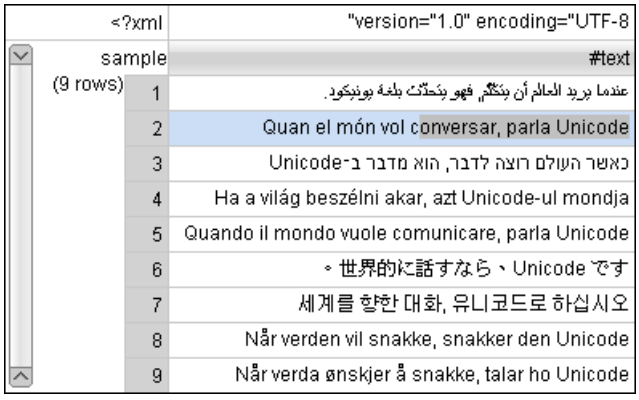


Figure 214: Right to left text orientation

Chapter 10

Transforming Documents

Topics:

- *Output Formats*
- *Transformation Scenario*
- *XSLT Processors*
- *XSL-FO Processors*
- *XProc Transformations*

XML is designed to store, carry, and exchange data, not to display data. When you want to view the data, you must either have an XML-compliant user agent or transform it to a format that can be read by other user agents. This process is known as transformation.

Status messages generated during transformation are displayed in the ***Information*** *view*.

Output Formats

Within the current version of Oxygen you can transform your XML documents to the following formats without having to exit from the application. For transformation to formats not listed simply install the tool chain required to perform the transformation and process the xml files created with Oxygen in accordance with the processor instructions.

- **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.

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 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 9.3.0.5 HE, Saxon 9.3.0.5 PE, and Saxon 9.3.0.5 EE. Also *the validation* is done in function of the stylesheet version.

Transformation Scenario

Before transforming an XML document in Oxygen you must 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 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 will run 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 will execute a DITA-OT build script. Oxygen 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.

A scenario can be created at *document type* level or at global level. The scenarios defined at document type level are available only for the documents that match that document type. The global scenarios are available for any document.

In order to apply a transformation scenario one has to press the  **Apply Transformation Scenario** button from the **Transformation** toolbar.

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:

-  **Apply Transformation Scenario** - Applies to each selected file the transformation scenario associated to that file. If the currently processed file does not have an associated transformation scenario then a warning is displayed in the **Warnings** view to let the user know about it.
- **Transform with...** - allows the user to select one transformation scenario to be applied to each one of the currently selected files.

Built-in Transformation Scenarios

If the **Apply Transformation Scenario** button from the **Transformation** toolbar is pressed, currently there is no scenario associated with the edited document and the edited document contains a `xml-stylesheet` processing instruction referring to a XSLT stylesheet (commonly used for display in Internet browsers), then Oxygen will prompt the user and offer the option to associate the document with a default scenario. The default scenario contains in the **XSL URL** field

the URL from the `href` attribute of the processing instruction. This scenario will have the **Use xml-stylesheet declaration** checkbox set by default, will use Saxon as transformation engine, will perform no FO processing and will store the result in a file with the same URL as the edited document except the extension which will be changed to `html`. The name and path will be preserved because the output file name is specified with the help of two *editor variables*: `${cfd}` and `${cfn}`.

Oxygen comes with preconfigured built-in scenarios for usual transformations that enable the user to obtain quickly the desired output: associate one of the built-in scenarios with the current edited document and then apply the scenario with just one click.

Defining a New Transformation Scenario

The **Configure Transformation Scenario** dialog is used to associate a scenario from the list of all scenarios with the edited document by selecting an entry from the list. The dialog is opened by pressing the **Configure Transformation Scenario** button on the **Transformation** toolbar of the document view. Once selected the scenario will be applied with only one click on the **Apply Transformation Scenario** on the same toolbar. Pressing the **Apply Transformation Scenario** button before associating a scenario with the edited document will invoke first the **Configure Transformation Scenario** dialog and then apply the selected scenario.

If there is only one scenario that can be associated with edited document, **Configure Transformation Scenario** dialog is not displayed, scenario is associated with edited document and transformation is executed. This situation can appear both for project scenarios and global scenarios when there is only one scenario in list of possible scenarios that include document type scenarios and project/global scenarios. Association of that scenario can be changed by opening **Configure Transformation Scenario** dialog with an action having the same name (**Configure Transformation Scenario**) from the toolbar.

Open the **Configure Transformation Scenario** dialog using one of the methods previously presented or by selecting **Document** > **Transformation** > **Configure transformation scenario. (Ctrl+Shift+C)** .

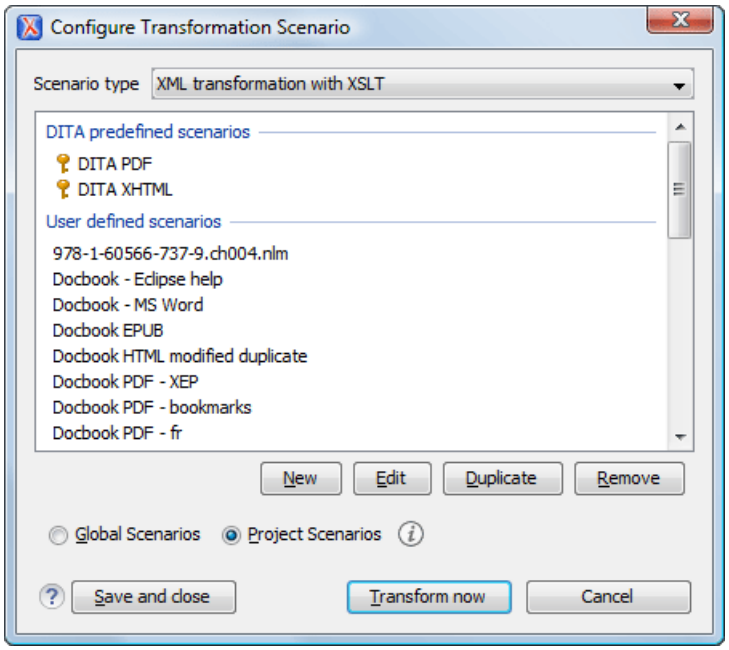


Figure 215: Configure Transformation Scenario Dialog

The **Scenario type** allows you to choose what type of user defined transformation scenario is displayed:

- **All** - No filtering. All user-defined scenarios are displayed.
- **XML transformation with XSLT** - Transformation scenarios that apply an XSLT stylesheet over an XML.
- **XML transformation with XQuery** - Transformation scenarios that apply an XQuery over an XML.

- **DITA OT transformation** - Transformation scenarios that use the DITA Open Toolkit (DITA-OT) to transform XML content into an output format.
- **ANT transformation** - Transformation scenarios that execute ANT scripts.
- **XSLT transformation** - Transformation scenarios that apply an XSLT stylesheet over an XML file.
- **XProc transformation** - Transformation scenarios that execute XProc XML pipelines.
- **XQuery transformation** - Represents a transformation that consists in applying an XQuery over an XML.
- **SQL transformation** - Executes an SQL over a database.

If you want an XSLT scenario select as **Scenario type** either **XML transformation with XSLT** or **XSLT transformation** then complete the dialog as follows:

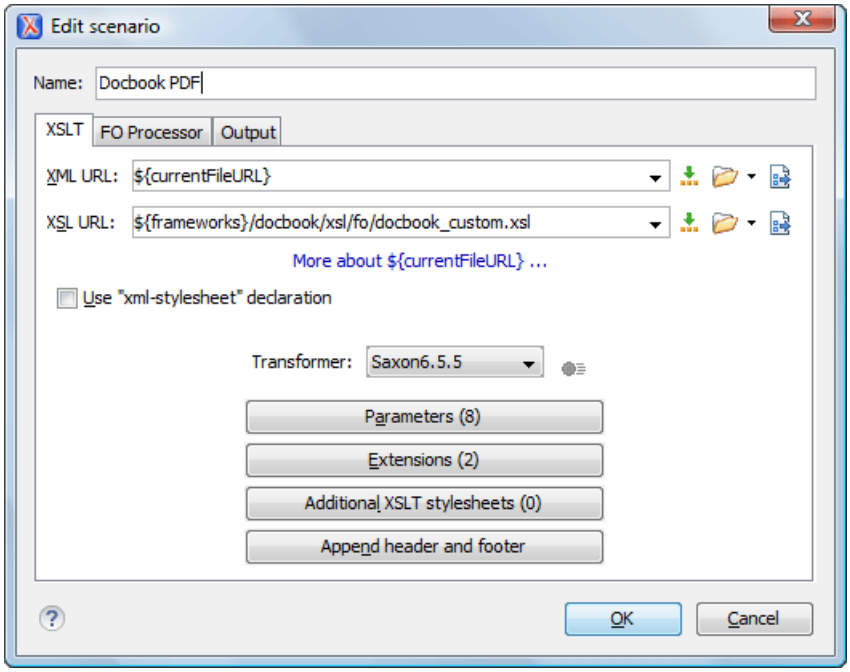


Figure 216: The Configure Transformation Dialog - XSLT Tab

- **XML URL** - Specifies an input XML file to be used for the transformation. Please note that this URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the editor will try to use the file directly.

  Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured for Saxon 9 in **Preferences** then the XML input of the transformation is passed to that URI resolver.

  Note: If the transformer engine is one of the built-in XSLT 2.0 engines and *the name of an initial template is specified in the scenario* then the **XML URL** field can be empty. Also the **XML URL** field can be empty in case of *external XSLT processors*. In all other cases a non-empty XML URL value is mandatory.

  The following buttons are shown immediately after the input field:

  - **Insert Editor Variables** - Opens a pop-up menu allowing to introduce special *Oxygen editor variables* or *custom editor variables* in the XML URL field.
  - **Browse for local file** - Opens a local file browser dialog allowing to select a local file name for the text field.
  - **Browse for remote file** - Opens a URL browser dialog allowing to select a remote file name for the text field.
  - **Browse for archived file** - Opens a zip archive browser dialog allowing to select a file name from a zip archive that will be inserted in the text field.
  - **Open in editor** - Opens the file with the path specified in the text field in an editor panel.

- **XSL URL** - Specifies an input XSL file to be used for the transformation. Please note that this URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the editor will try to use the file directly. The above set of browsing buttons are available also for this input.
- **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 contains all the transformer engines available for applying the stylesheet. These are the built-in engines and *the external engines defined in the user preferences*. If you want to change the default selected engine just select other engine from the drop down list of the combo box. For XSLT/XQuery files only, if no validation scenario is associated, the transformer engine will be used in validation process, if it has validation support.
- **Parameters** - Opens *the dialog for configuring the XSLT parameters.* In this dialog you set any global XSLT parameters of the main stylesheet set in the **XSL URL** field or of the additional stylesheets set with the button **Additional XSLT stylesheets**. If the XSLT transformer engine is custom defined this dialog cannot be used 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 in the command line the necessary parameters.
- **Append header and footer** - Opens a dialog for specifying a URL for a header HTML file added at the beginning of the result of an HTML transformation and a URL for a footer HTML file added at the end of the HTML result of 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.
- **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 XSLT/XQuery transformation.
- **Advanced options** - Configure advanced options specific for the Saxon HE / PE / EE 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. By default if you do not set a specific value in the transformation scenario each advanced option has the same value as the global option with the same name *set in the user preferences*.

  The advanced options include two options that are not available globally in the user preferences: the initial XSLT template and the initial XSLT mode of the transformation. They are Saxon specific options that allow imposing the name of the first XSLT template that starts the XSLT transformation or the initial mode of transformation.

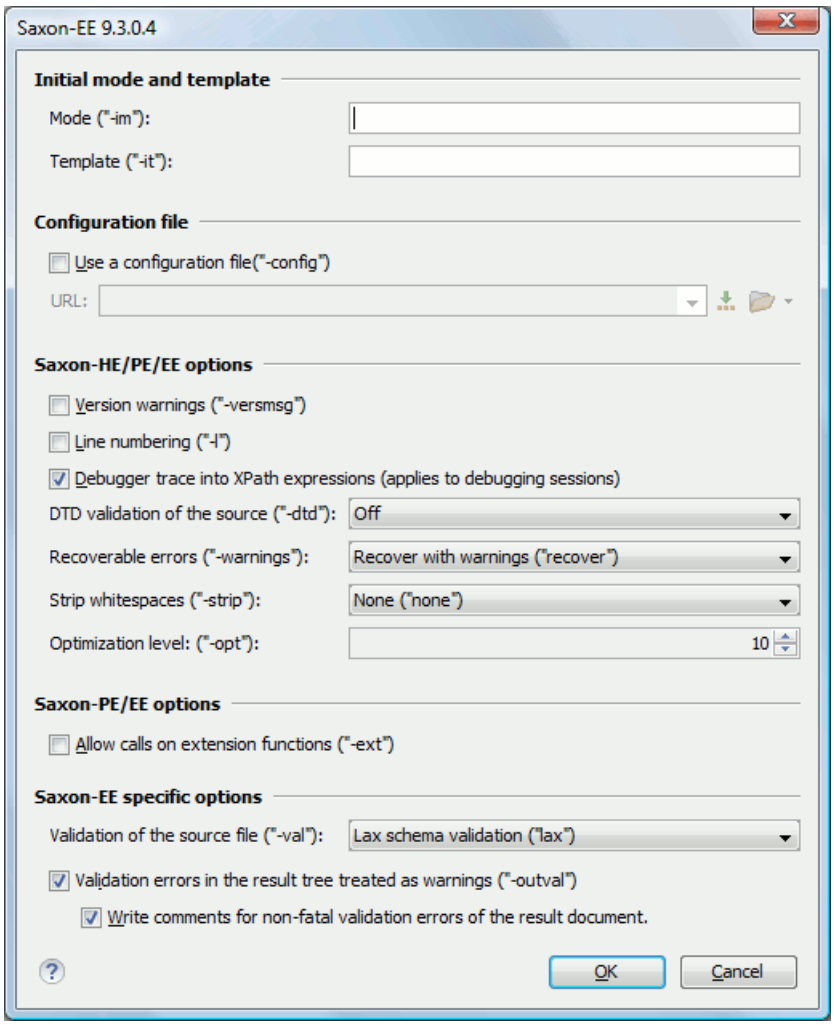


Figure 217: The advanced options of Saxon HE / PE / EE

The advanced options specific for Saxon HE / PE / EE are:

- **Mode ("-im")** - Specifies to the transformer the initial template mode
- **Template ("-it")** - Specifies the name of the initial template to the transformer. When specified, the XML input URL for the transformation scenario is optional.
- **Use a configuration file ("-config")** - The **URL** input points to Saxon advanced options configuration file.
- **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.
- **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 error as non-fatal if it fails

  Note that 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. Either 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. Strips no whitespace before further processing. However, whitespace will still be 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 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, 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 thereby gain privileged access to resources on your machine.

The advanced options available only in Saxon EE are:

- **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 determines whether source documents should be parsed with schema-validation enabled.
    - **Lax schema validation ("lax")** - This mode determines whether source documents should be parsed with schema-validation enabled if an XML Schema is provided.
    - **Disable schema validation** - This determines whether 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.

When creating a scenario that applies to an XML file, Oxygen fills the **XML URL** field with the default variable *${currentFileURL}*. This means the input for the transformation is taken from the currently edited file. You can modify this value to other file path. This is the case of currently editing a section from a large document, and you want the transformation to be performed on the main document, not the section. You can specify in this case either a full absolute path: `file:/c:/project/docbook/test.xml` or a path relative to one of the editor variables, like the current project file: `${pdu}/docbook/test.xml` .

When the scenario applies to XSL files, the field `XSL URL` is containing *${currentFile}*. Just like in the XML case, you can specify here the path to a master stylesheet. The path can be configured using the *editor variables* or the *custom editor variable*.

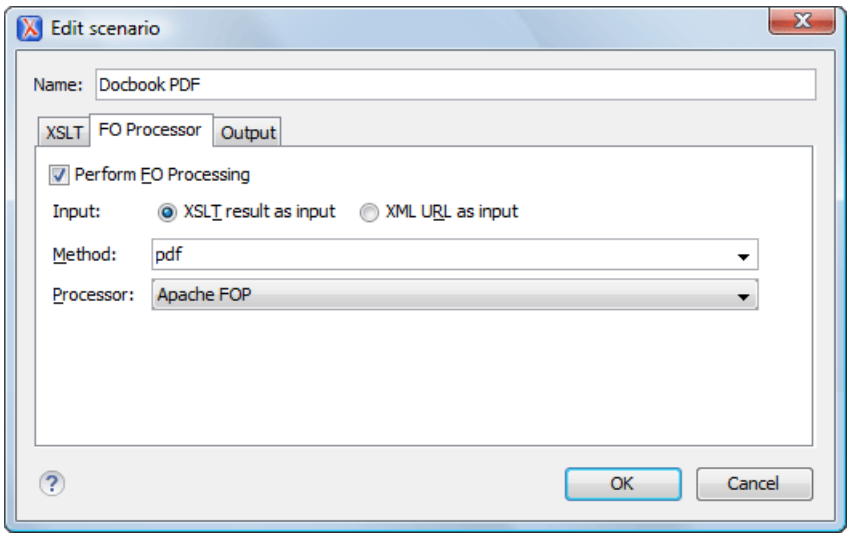


Figure 218: The Configure Transformation Dialog - FO Processor Tab

- **Perform FO Processing** - Enables or disables applying an FO processor (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 on the XSLT tab of the dialog.
- **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** - The FO processor, which can be the built-in Apache FOP processor or an *external processor.*

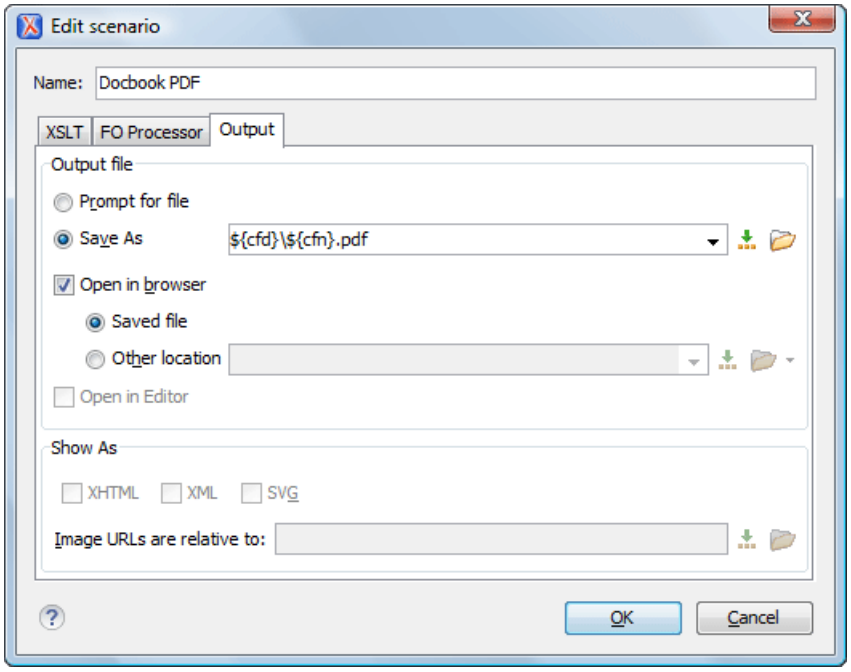


Figure 219: The Configure Transformation Dialog - Output Tab

- **Prompt for file** - At the end of the transformation a file browser dialog will be displayed for specifying the path and name of the file which will store the transformation result.
- **Save As** - The path of the file where it will be stored the transformation result. The path can include *special Oxygen editor variables* or *custom editor variables*.

- **Open in browser** - If this is checked Oxygen will open automatically the transformation result in a browser application specific for the type of that result (HTML/XHTML, PDF, text).
- **Saved file** - When **Open in browser** is selected this button can be selected to specify that Oxygen should open automatically at the end of the transformation the file specified in the **Save As** text field.
- **Other location** - When **Open in browser** is selected this button can be used to specify that Oxygen 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 editor variables* or *custom editor variable*.
- **Open in editor** - When this is checked the transformation result set in the **Save As** field is opened in a new editor panel in Oxygen 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, etc.
- **Show As XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked Oxygen will display the transformation result in a built-in XHTML browser panel at the bottom of the Oxygen window.

  Important: When transforming very large documents you should be aware that enabling this feature will result 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 this situations if you wish to see the XHTML result of the transformation you should use an external browser by checking the **Open in browser** checkbox.

- **Show As XML** - If this is checked Oxygen will display the transformation result in an XML viewer panel at the bottom of the Oxygen window with *syntax highlight* specific for XML documents.
- **Show As SVG** - If this is checked Oxygen will display the transformation result in a SVG viewer panel at the bottom of the Oxygen window by rendering the result as a SVG image.
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path for resolving image paths contained in the transformation result.

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:

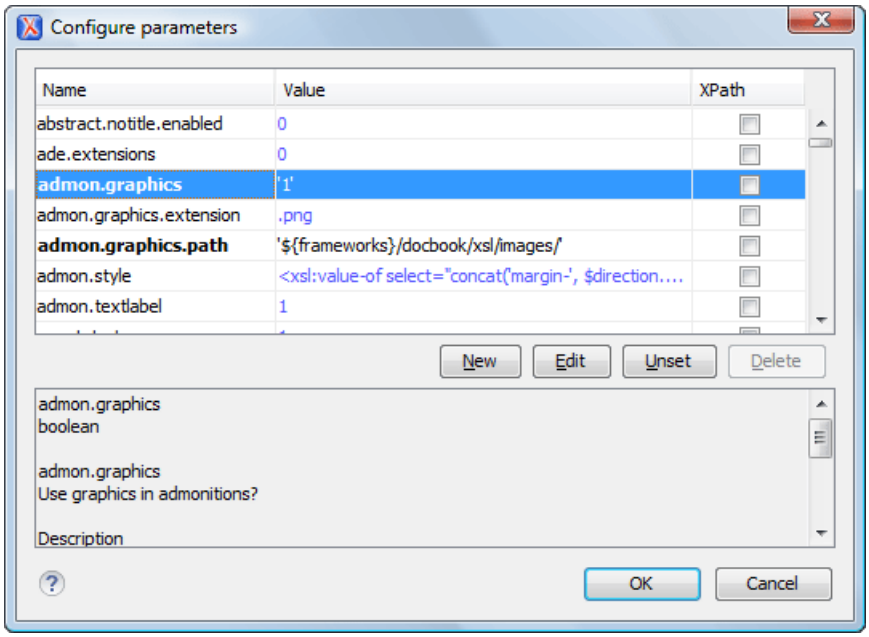


Figure 220: Configure parameters dialog

The table presents all the parameters of the XSLT stylesheet, all imported and included stylesheets and all *additional stylesheets* with their current values. 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 it is available and the system ID of the stylesheet that declares it.

For setting the value of a parameter having a namespace, for example 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 the 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 (*${frameworks}*, *${home}*, *${cfd}*, etc) can be used in the values of the parameters to make the value independent of the location of the XSLT stylesheet or the XML document.

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:

- `${ask('message')}` - Only the message displayed for the user is specified.
- `${ask('message', generic, 'default')}` - `'message'` will be displayed for the user, the type is not specified (the default is string), the default value will be `'default'`.
- `${ask('message', password)}` - `'message'` will be displayed for the user, the characters typed will be replaced with a circle character.
- `${ask('message', password, 'default')}` - Same as above, default value will be `'default'`.
- `${ask('message', url)}` - `'message'` will be displayed for the user, the type of parameter will be URL.
- `${ask('message', url, 'default')}` - Same as above, default value will be `'default'`.

Additional XSLT Stylesheets

The list of additional XSLT stylesheets can be edited in the dialog opened by the button **Additional XSLT Stylesheets**.

- **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.
- **New** - Opens a dialog in which you can type the name of a stylesheet. The name is considered relative to the URL of the current edited XML document. You can use *editor variables* in the name of the stylesheet. The name of the stylesheet will be added in the list after the current selection.

- **Remove** - Deletes the selected stylesheet from the **Additional XSLT stylesheets** list.
- **Open** - Opens the selected stylesheet in a separate view.
- **Up** - Moves the selected stylesheet up in the list.
- **Down** - Moves the selected stylesheet down in the list.

This dialog allows the user to add additional XSLT stylesheets to the transformation.

The path specified in the URL text field can include *special Oxygen editor variables.*

XSLT/XQuery Extensions

The **Edit Extensions** 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.

Creating a Transformation Scenario

Use the following procedure to create a transformation scenario.

1. Go to menu **Document** > **Transformation** > **Configure Transformation Scenario (Ctrl+Shift+C)** 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 in the **Name** field and type a new name.
4. Click **OK** or **Transform Now** to save the scenario.

ANT Transformations

The following options are available in the **Options** tab:

- **Working directory** - Path of the directory where results are stored.
- **Build file** - ANT script file.
- **Build target** - You can specify a build target to the ANT script file. By default no target is necessary and the default *init* target is used.
- **Additional arguments** - Additional command-line arguments to be passed to the ANT transformation (for example -verbose).
- **Ant Home** - Path to the custom ANT installation to run the transformation. By default it is the ANT installation bundled with Oxygen.
- **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.
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to -Xmx256m which means the transformation process is allowed to use 256 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 (256 MB) to a higher value, like 512 MB. In this way, you can avoid running out of memory (**OutOfMemoryError**) when running and ANT process.

In the **Parameters** tab use the toolbar buttons to add, edit or remove transformation parameters.

Use the **Output** tab to set the file to open after the transformation has finished. Also you can choose what application is used for opening the output file.

Sharing the Transformation Scenarios

The transformation scenarios can be shared with other users by saving them at project level. In the upper part of the dialog showing the list of scenarios you will find two radio buttons controlling where the scenarios are stored: **Global Scenarios** and **Project Scenarios**.

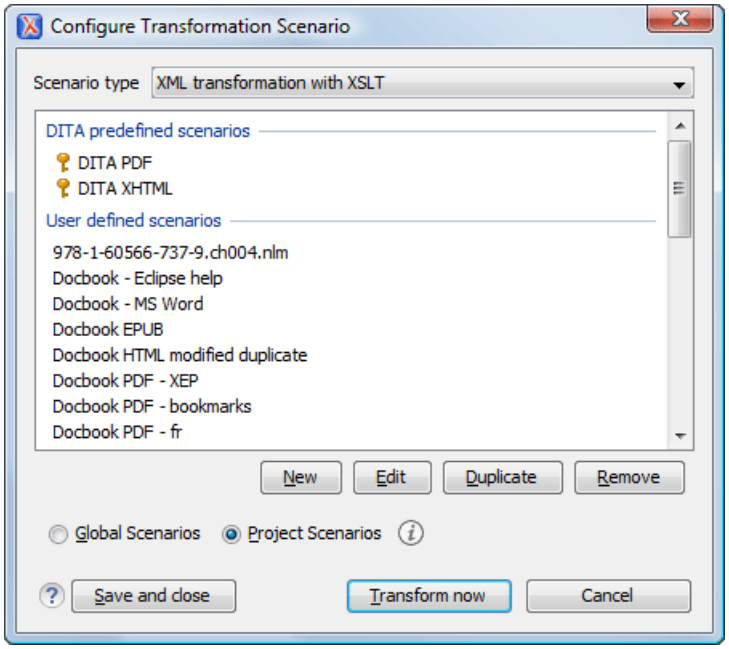


Figure 221: Transformation Scenario List Dialog

Selecting **Global Scenarios** ensures that the scenarios are saved in the user home directory.

After changing the selection to **Project Scenarios**, the scenario list will be stored in the project file. If your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc..) then your team can use the scenarios you defined.

Predefined scenarios are presented according to the current document's detected type. The screenshot above shows all default scenarios for a *DocBook 4* document and one custom transformation scenario. The key symbol before the scenario name indicates that the scenario can only be modified from the ***Document Type Association*** options page.

Other preferences can also be stored at the project level. For more information, see the *Preference Sharing* section.

Transformation Scenarios View

The list of transformation scenarios may be easier to manage for some users as a list presented in a dockable and floating view called **Transformation Scenarios**.

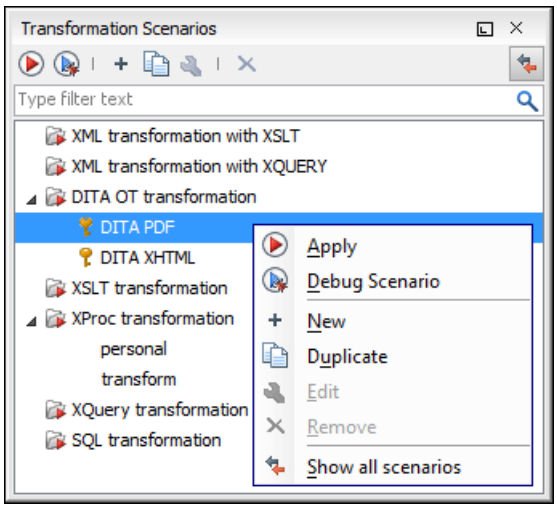


Figure 222: The Scenarios view

The actions available on the right click menu allow the same operations as in *the dialog* ***Configure Transformation Scenario***:

- **Apply** - Runs the current transformation scenario that is selected in the list of scenarios.
- **Debug Scenario** - Switches 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.
- **New** - Creates a new transformation scenario.
- **Duplicate** - Adds a new scenario to the list that is a duplicate of the current scenario. It is useful for creating a new 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.
- **Show all scenarios / Show current editor scenarios** - A toggle action that switches between two modes: show / hide the scenarios that are specified in the *document type* corresponding to the current editor. All global scenarios (the scenarios that are not specified in a document type) are always displayed in the view regardless of the state of this action.

XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen.

Supported XSLT Processors

Oxygen 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 9.3.0.5 Home Edition (HE), Professional Edition (PE)** - *Saxon-HE/PE* implements the basic conformance level for XSLT 2.0 and XQuery 1.0. The term *basic XSLT 2.0 processor* is defined in the draft XSLT 2.0 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 9.3.0.5 Enterprise Edition (EE)** - *Saxon EE* is the schema-aware edition of Saxon and it is one of the built-in processors of Oxygen. 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 specification or according to the W3C XML Schema 1.1 one. This can be *configured in Preferences*.

Besides the above list Oxygen 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 is 2.7.6 and the libxslt version is 1.1.26

  Oxygen 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.

  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 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 use 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 get an 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 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 is 1.6.

  You should have the .NET Framework version 1.0 already installed on your system otherwise you get this 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 get this 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 get this 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*

The button **Transformation options** available on the **Transformation** toolbar allows quick access to the *XSLT options* in the Oxygen user preferences.

Configuring Custom XSLT Processors

You can *configure other XSLT transformation engines* than *the ones which come with the Oxygen distribution*. Such an external engine can be used for XSLT transformations within Oxygen, in the Editor perspective, and is available in the list of engines in *the dialog for editing transformation scenarios* . However it cannot be used in the XSLT Debugger perspective.

The output messages of a custom processor are displayed in an output view at the bottom of the Oxygen window. If an output message follows *the format of an Oxygen 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.

The DocBook extensions for Xalan and Saxon are included in the `[Oxygen-install-directory]\frameworks\docbook\xsl\extensions` folder.

Samples on how to use extensions can be found at:

- for Xalan - *http://xml.apache.org/xalan-j/extensions.html*
- for Saxon 6.5.5 - *http://saxon.sourceforge.net/saxon6.5.5/extensions.html*
- for Saxon 9.3.0.5 - *http://www.saxonica.com/documentation/extensibility/intro.xml*

In order to set an XSLT processor extension (a directory or a jar file), you have to 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.

The Built-in XSL-FO Processor

The Oxygen 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 TIFF images you need the *JAI* library. For PDF images you need the *`fop-pdf-images`* library. These libraries are not bundled with Oxygen. Using them is as easy as downloading them and *creating a external FO processor* based on the built-in FOP libraries and the extension library. The *external FO processor created in* ***Preferences*** will have a command line like:

```
java -cp "${oxygenInstallDir}/lib/xercesImpl.jar:
${oxygenInstallDir}/lib/fop.jar:${oxygenInstallDir}/lib/
avalon-framework-4.2.0.jar:
${oxygenInstallDir}/lib/batik-all-1.7.jar:${oxygenInstallDir}/lib/
```

```
commons-io-1.3.1.jar:
${oxygenInstallDir}/lib/xmlgraphics-commons-1.3.1.jar:
${oxygenInstallDir}/lib/commons-logging-1.0.4.jar:
${oxygenInstallDir}/lib/saxon9ee.jar:${oxygenInstallDir}/lib/
saxon9-dom.jar:
${oxygenInstallDir}/lib/xalan.jar:${oxygenInstallDir}/lib/
serializer.jar:
${oxygenInstallDir}/lib/resolver.jar:${oxygenInstallDir}/lib/
fop-pdf-images-1.3.jar:
${oxygenInstallDir}/lib/PDFBox-0.7.3.jar"
org.apache.fop.cli.Main -fo ${fo} -${method} ${out}
```

You need to add to the classpath `JimiProClasses.zip` for *JIMI* and `jai_core.jar`, `jai_codec.jar` and `mlibwrapper_jai.jar` for *JAI*. For the *JAI* package you can include the directory containing the native libraries (`mlib_jai.dll` and `mlib_jai_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/>
      </fonts>
    </renderer>
  </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 using DITA-OT you should use an IDIOM FOP transformation and modify the following two files:
     - `${frameworks}/dita/DITA-OT/demo/fo/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/demo/fo/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 the special font that must be set to Apache FOP is installed in the operating system there is a simple way of telling FOP to look for the font. See *the simplified procedure for adding a font to FOP*.

1. Locate the font.

   First, you have to find out the name of a font that has the glyphs for the special characters you used. One font that covers the majority of 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.
   a) Open a terminal.
   b) Change the working directory to the Oxygen install directory.
   c) Create the following script file in the Oxygen installation directory.

   For Mac OS X and Linux create a file `ttfConvert.sh`:

```
#!/bin/sh
export LIB=lib
export CMD=java -cp
"$LIB/fop.jar:$LIB/avalon-framework-4.2.0.jar:$LIB/xercesImpl.jar"
export CMD=$CMD org.apache.fop.fonts.apps.TTFReader
export FONT_DIR='/Library/Fonts'
$CMD $FONT_DIR/Arialuni.ttf Arialuni.xml
```

   For Windows create a file `ttfConvert.bat`:

```
set LIB=lib
set CMD=java -cp
"%LIB%\fop.jar;%LIB%\avalon-framework-4.2.0.jar;%LIB%\xercesImpl.jar"
set CMD=%CMD% org.apache.fop.fonts.apps.TTFReader
set FONT_DIR=C:\Windows\Fonts
%CMD% %FONT_DIR%\Arialuni.ttf Arialuni.xml
```

   The relative paths specified in the file are relative to the Oxygen installation directory so if you decide to create it in other directory you have to change the file paths.

   The *FONT_DIR* can be different on your system. Make sure it points to the correct font directory. If the Java executable is not in the *PATH* you will have to specify the full path of the executable.

   If the font has bold and italic variants, you will have to convert those too. For this you add 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 you should execute the command `sh ttfConvert.sh` from the command line. On Windows you should run the command `ttfConvert.bat` from the command line or double click on the file `ttfConvert.bat`.

3. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)

a) Create a FOP configuration file that specifies the font metrics file for your font.

```
<fop version="1.0">
  <base>./</base>
  <font-base>file:/C:/path/to/FOP/font/metrics/files/</font-base>
  <source-resolution>72</source-resolution>
  <target-resolution>72</target-resolution>
  <default-page-settings height="11in" width="8.26in"/>
  <renderers>
    <renderer mime="application/pdf">
      <filterList>
        <value>flate</value>
      </filterList>
      <fonts>
          <font metrics-url="Arialuni.xml" kerning="yes"
                embed-url="file:/Library/Fonts/Arialuni.ttf">
              <font-triplet name="Arialuni" style="normal"
                   weight="normal"/>
          </font>
      </fonts>
    </renderer>
  </renderers>
</fop>
```

The `embed-url` attribute points to the font file to be embedded. You have to 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">
  ...
    <fonts>
        <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 *http://xmlgraphics.apache.org/fop/0.93/configuration.html*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 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 **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 transformations using DITA-OT you should use an IDIOM FOP transformation and modify the following two files:

- `${frameworks}/dita/DITA-OT/demo/fo/cfg/fo/font-mappings.xml` - the `font-face` element included in each element `physical-font` having the `attribute char-set="default"` must contain the name of the font (*Arialuni* in our example)
- `${frameworks}/dita/DITA-OT/demo/fo/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>
```

XProc Transformations

This section explains how to configure and run XProc transformations in Oxygen.

XProc Transformation Scenario

A sequence of transformations described by an XProc script can be executed with an XProc transformation scenario. In the scenario the parameters of the transformation are specified:

- the URL of the XProc script
- the XProc engine
- the input ports
- the output ports

On the **XProc** tab of the scenario edit dialog it is selected the URL of the XProc script and the XProc engine. The engine can be the built-in one (*Calabash*) or a custom engine *configured in the **Preferences** dialog*.

On the **Inputs** tab of the dialog is configured each port that is used in the XProc script for reading input data. Each input port has a name that is assigned in the XProc script and that is used for identifying the port in the list from the **Port** combo box. The XProc engine will read data from the URLs specified in the **URLs** list. The *built-in editor variables* and the *custom editor variables* can be used for specifying a URL.

On the *Parameters* tab you can specify the parameters available on each port.

Each port where is sent the output of the XProc transformation is associated with a URL on the **Outputs** tab of the dialog. The *built-in editor variables* and the *custom editor variables* can be used for specifying a URL.

The result of the XProc transformation can be displayed as a sequence in an output view with two sides: a list with the output ports on the left side and the content of the document(s) that correspond to the output port selected on the left side. If the option **Open in editor** is selected, the XProc transformation result will be opened automatically in an editor panel. If the option **Open in browser** is selected, you can specify a file to be opened in the default browser at the end of the XProc transformation.

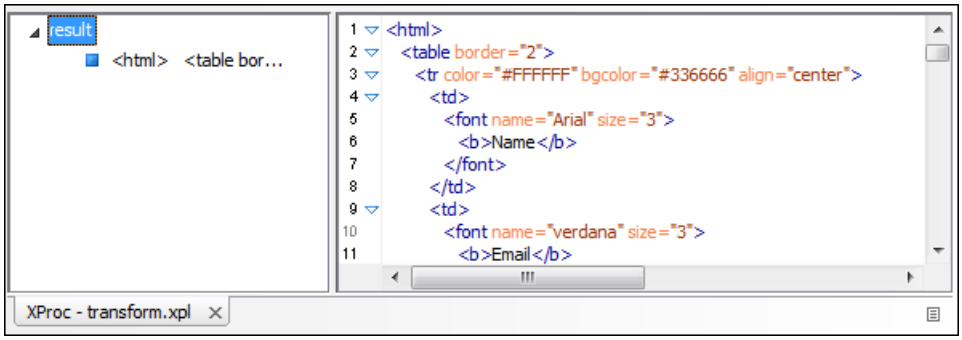


Figure 223: XProc Transformation results view

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download in the following zip file: *`xprocAPI.zip`*. In order to create an XProc integration project the following requirements must be fulfilled:

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. The Javadoc documentation for the XProc API is available on our website: *`xprocAPI.zip`*.
3. Create a new Java archive (jar) from the classes you created.
4. Create a new `engine.xml` file according with the `engine.dtd` file. The attributes of the `engine` tag 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` tag has only one child, `runtime`. The `runtime` tag contains several `library` elements who's attribute `name` contains the relative or absolute location of the libraries necessary to run this integration.
5. Create a new folder with the name of the integration in the `[Oxygen-install-folder]/lib/xproc`.
6. Put there the `engine.xml`, and all the libraries necessary to run properly the new integration.

Chapter 11

Querying Documents

Topics:

- *Running XPath Expressions*
- *Working with XQuery*

This chapter shows how to query XML documents in Oxygen with XPath expressions and with the XQuery language.

Running XPath Expressions

This section explains possible ways of running an XPath expression on an XML document.

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 a Structured Query Language (SQL) query used to select records from a database.

XPath models an XML document as a tree of nodes. 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.

Some examples:

- `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, the following URL is recommended: *http://www.w3.org/TR/xpath*.

Oxygen's XPath Console

To use XPath effectively requires at least an understanding of *the XPath Core Function Library*. If you have this knowledge the Oxygen XPath expression field part of the current editor toolbar can be used to aid you in XML document development.

In Oxygen a XPath 1.0 or XPath 2.0 expression is typed and executed on the current document from the XPath console available on the XPath toolbar for every open XML document.. Both XPath 2.0 basic and XPath 2.0 schema aware expressions can be executed in the XPath console. XPath 2.0 schema aware also takes into account the Saxon EE *XML Schema version* option.

The *content completion assistant* that helps in entering XPath expressions in attributes of XSLT stylesheets elements is also available in the XPath console and offers always proposals dependent of the current context of the cursor inside the edited document. The set of XPath functions proposed by the assistant depends on the XPath version selected from the drop-down menu of the XPath button (1.0 or 2.0).

In the following figure the cursor is on a `person` element and the content completion assistant offers all the child elements of the `person` element and all XPath 2.0 functions:

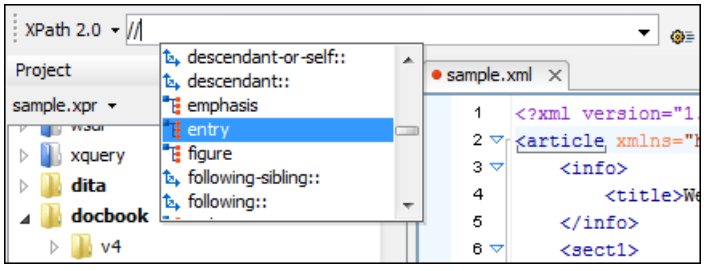


Figure 224: Content Completion in the XPath console

The evaluation of the XPath expression tries to resolve the locations of documents referred in the expression through the *XML catalogs* which are *configured in Preferences* and *the current XInclude preferences*. An example is evaluating

the `collection(URIofCollection)` function (XPath 2.0). If you need to resolve the references from the files returned by the `collection()` function with an XML catalog set up in the Oxygen preferences you have to specify the class name of the XML catalog enabled parser for parsing these collection files. The class name is `ro.sync.xml.parser.CatalogEnabledXMLReader` and you specify it like this:

```
let $docs := collection(iri-to-uri(
    "file:///D:/temp/test/XQuery-catalog/mydocsdir?recurse=yes;select=*.xml;
    parser=ro.sync.xml.parser.CatalogEnabledXMLReader"))
```

If you want to see in the XPath console the XPath expression at the current cursor position when navigating in the document you have to check the button **XPath update on caret move.**

The results of an XPath query are returned in the message panel. Clicking a record in the result list highlights the nodes within the text editor panel with a character level precision. Results are returned in a format that is a valid XPath expression:

```
- [FileName.xml] /node[value]/node[value]/node[value] -
```

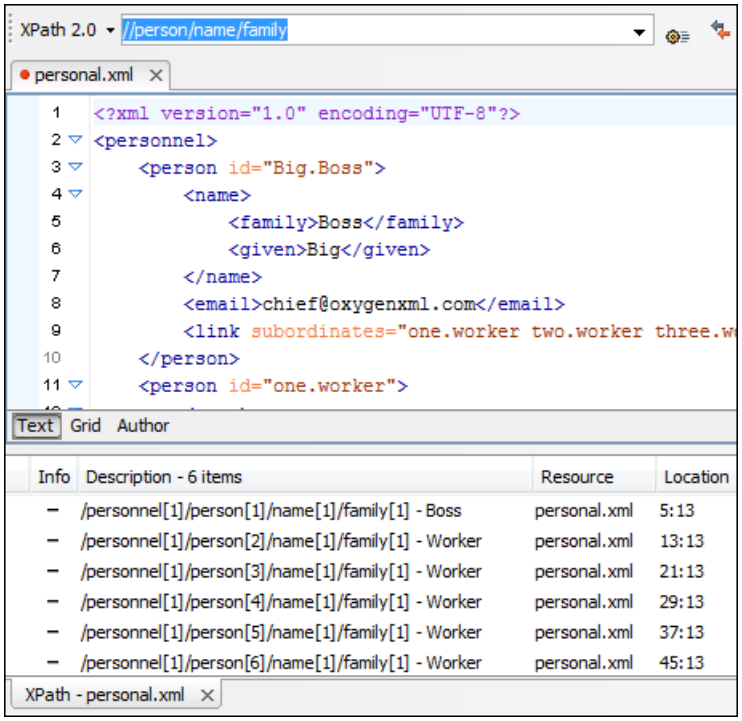


Figure 225: XPath results highlighted in editor panel with character precision

When using the *grid editor*, clicking a result record will highlight the entire node.

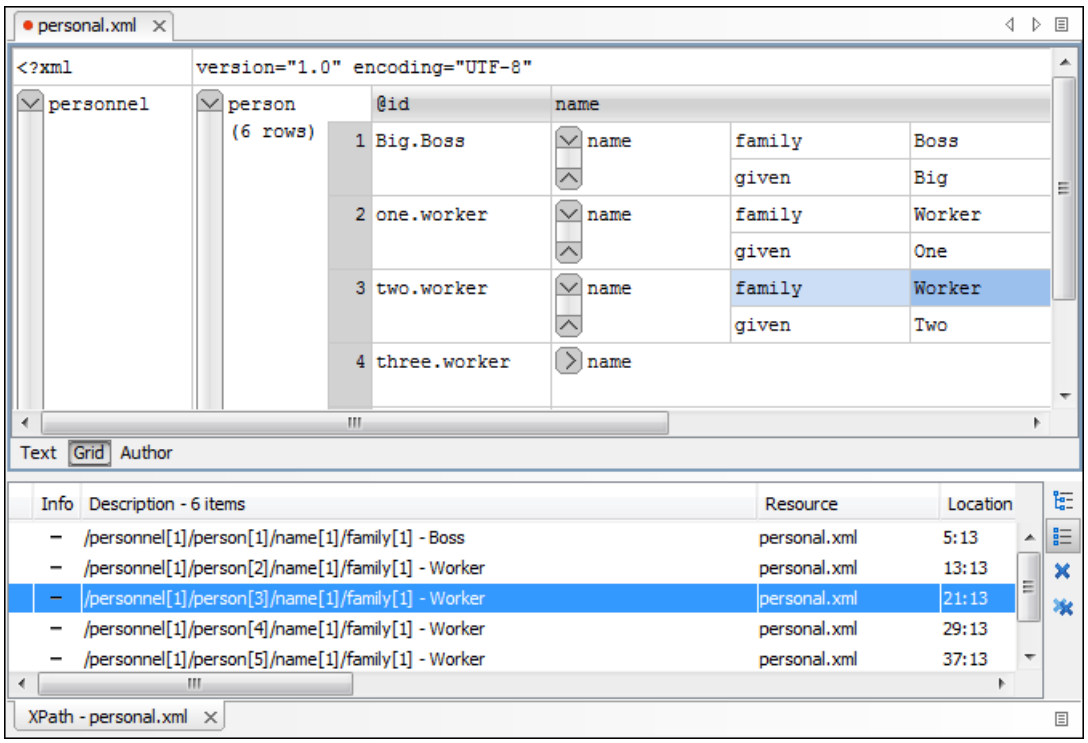


Figure 226: XPath results highlighted in the Grid Editor

When the limit of long expressions is reached (60 characters) a dialog pops up and offers to switch the focus to the *XPath builder* view. This is a view specially designed to assist you with typing and testing complex XPath 1.0 / 2.0 expressions.

XPath Utilization 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. DocBook defines that chapters have a `<chapter>` start tag and a matching `</chapter>` end tag to close the element. To return all the chapter nodes of the book you should enter `//chapter` into the XPath expression field then press **(Enter)**. This will return all the `chapter` nodes of the DocBook book, in the message panel. If your book has six chapters, they will be six records in the result list. Each record when clicked will locate and highlight the chapter and all sibling nodes contained between the start and end tags of the chapter.

If you want to find all `example` nodes contained in the `section 2` nodes of a DocBook XML document you should use the following XPath expression: `//chapter/sect1/sect2/example`. If an `example` node is found in any `section 2` node, a result will be returned to the message panel. For each occurrence of the element node a record will be created in the result list.

For example one of the results of the previous XPath query on the file `oxygen.xml` is:

```
- [oxygen.xml] /chapter[1]/sect1[3]/sect2[7]/example[1]
```

which means that in the file `oxygen.xml`, first chapter, third section level 1, seventh section level 2, the example node found is the first in the section.

Important: If the document defines a default namespace then Oxygen will bind this namespace to the first free prefix from the list: `default`, `default1`, `default2`, etc. For example if the document defines the default namespace `xmlns="something"` and the prefix `default` is not associated with a namespace then you can match tags without prefix in a XPath expression typed in the XPath console by using the prefix `default`. For example to find all the `level` elements when the root element defines a default namespace you should execute in the XPath console the expression: `//default:level`.

To define default mappings between prefixes that can be used in the XPath console and namespace URIs *go to the* ***XPath Options*** *user preferences panel* and enter the mappings in the **Default prefix-namespace mappings** table. The same preferences panel allows also the configuration of the default namespace used in XPath 2.0 expressions entered into the XPath toolbar and the creation of different message panels for XPath queries executed on different XML documents.

To apply a XPath expression relative to the element on which the caret is positioned use the following actions:

- **Document** > **XML Document** > **Copy XPath (Ctrl+Alt+.)** (also available on the context menu of the main editor panel) to copy the XPath expression of the current element or attribute to the clipboard
- the **Paste** action of the contextual menu of the XPath console to paste this expression in the console
- add your relative expression in the console and execute the resulting complete expression

The XPath Builder View

The `XPath Builder` view allows you can compose complex XPath expressions with the help of *XPath content completion assistant* and color syntax highlight support. The view is opened from **Window** > **Show View** > **XPath Builder** menu action.

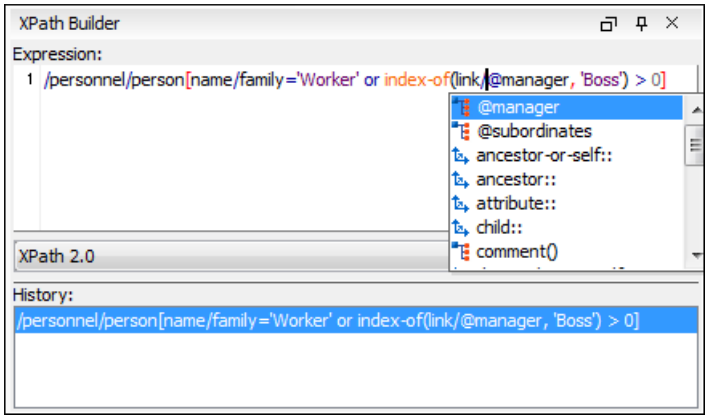


Figure 227: The XPath Builder View

The **Execute** button runs the expression against the edited document and takes into account the value selected in the combo box with the XPath version number: 1.0 or 2.0. Both XPath 2.0 basic and XPath 2.0 schema aware expressions can be evaluated. The *XPath preferences panel* is accessible from the **XPath Options** shortcut button near the **Execute** button. Oxygen keeps a history list of the XPath expressions executed in the current session, so you can reuse them to compose new ones.

The **XPath update on caret move** button enables the **XPath Builder** view to display the XPath expression at the current cursor position when navigating in the document.

The evaluation of the XPath expression tries to resolve the locations of documents referred in the expression through the *XML catalogs* which are *configured in Preferences* pages and *the current XInclude preferences*. For example, the preferences are used when evaluating the `collection(URIofCollection)` function (XPath 2.0).

The results of the XPath query are displayed in the same message panel as for *the XPath console* and are computed with the same *character level precision.*

The *usual edit actions* **Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo** are available in the popup menu of the top part of the view, where XPath expressions are edit. For the history list area of the view the popup menu contains two actions:

- **Execute** - Executes again the expression selected in the list.
- **Remove** - Removes the selected expression from the list.

Working with XQuery

This section explains how to edit and run XQuery queries in Oxygen.

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 a new XQuery document* select **File** > **New (Ctrl+N)** and when the **New** dialog appears select XQuery entry.

In the XQuery document Oxygen provides syntax highlight for keywords and all known XQuery functions and operators. Also for these there is available a content completion component that is activated with the **(Ctrl-Space)** shortcut. The functions and operators are presented together with a comment about parameters and functionality. For some supported database engines like eXist and Berkeley DB, the content completion lists contain the XQuery functions implemented by that engine. The XQuery file must have an associated transformation scenario which uses one of the specified engines or the validation uses one of these engines.

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 9.3.0.5 PE or Saxon 9.3.0.5 EE
- the edited file has a validation scenario associated that use as validation engine Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE
- the validation engine specified in *Preferences* is Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE.

If the Saxon namespace (*http://saxon.sf.net*) is mapped to a prefix this prefix is used when the functions are presented, otherwise the default prefix for the Saxon namespace (`saxon`) is used.

If you want to use a function from a namespace mapped to a prefix, just type that prefix and the content completion will display all the XQuery functions from that namespace. The XQuery functions from default namespace offered by content completion are prefixed if the default namespace is mapped to a prefix, otherwise is displayed just the name of this.

The content completion popup window presents all the variables and functions from both the edited XQuery file and its imports.

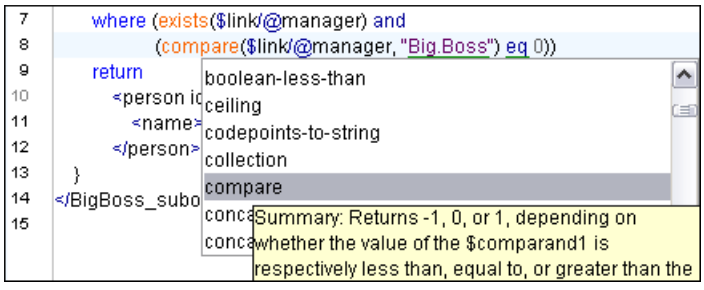


Figure 228: XQuery Content Completion

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 a quick access to a component by knowing its name. It is opened from menu **Window** > **Show View** > **Outline** .

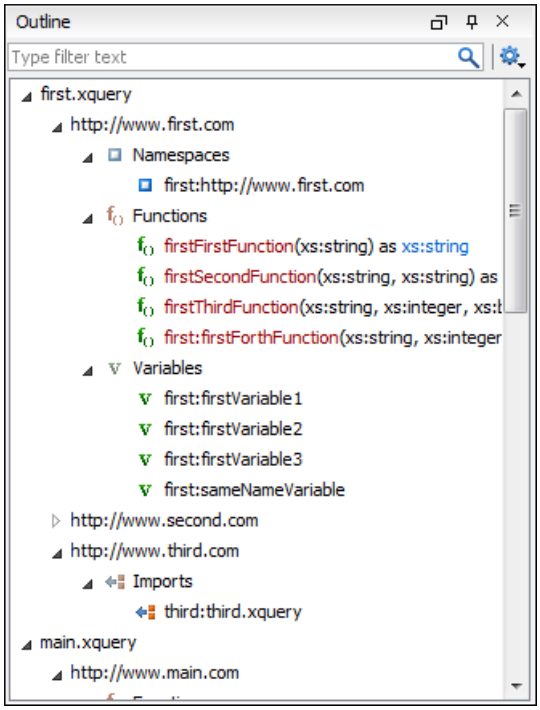


Figure 229: XQuery Outline View

The following actions are available in the **Settings** menu on the Outline view's toolbar:

- **Selection update on caret move** - Allows a synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret's moves or the changes 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 sort alphabetically the XQuery components.
- **Show all components** - Displays all components that were collected starting from the current file. This option is set by default.
- **Show local components** - Displays 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 the first presented 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 bottom 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 searched 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

A node can be dragged and dropped in the editor area for quickly inserting `doc()` or other XQuery expressions.

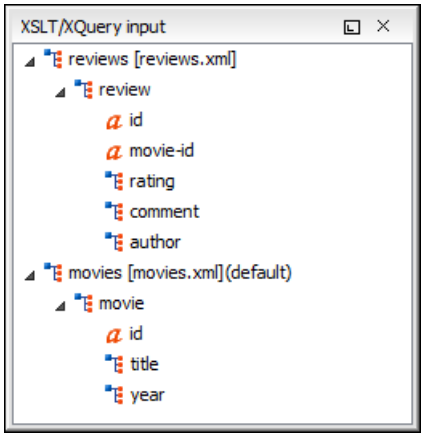


Figure 230: XQuery Input view

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 **rating** element and drop between the braces a popup menu will be displayed.

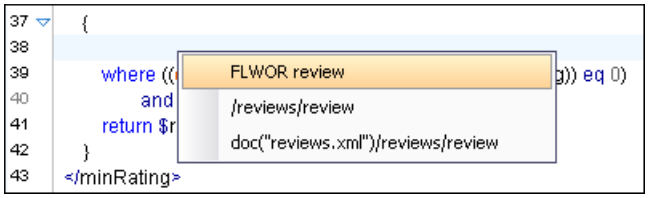


Figure 231: XQuery Input drag and drop popup menu

Select **FLWOR rating** and the result document will be:

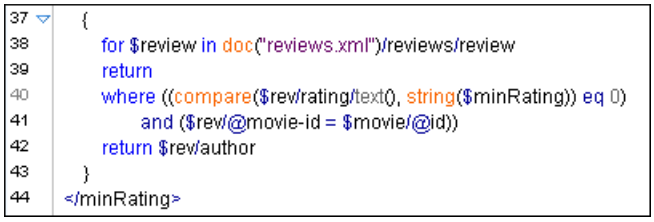


Figure 232: XQuery Input drag and drop result

XQuery Validation

With Oxygen you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.3.0.5 PE processor or the 9.3.0.5 EE, IBM DB2, eXist, Software AG Tamino, Berkeley DB XML or Documentum xDb (X-Hive/DB) 10 if you installed them. Also any XQuery processor that offers an *XQJ API implementation* can 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 syntactically check the expression 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 on one entry, the line where the error appeared is highlighted.

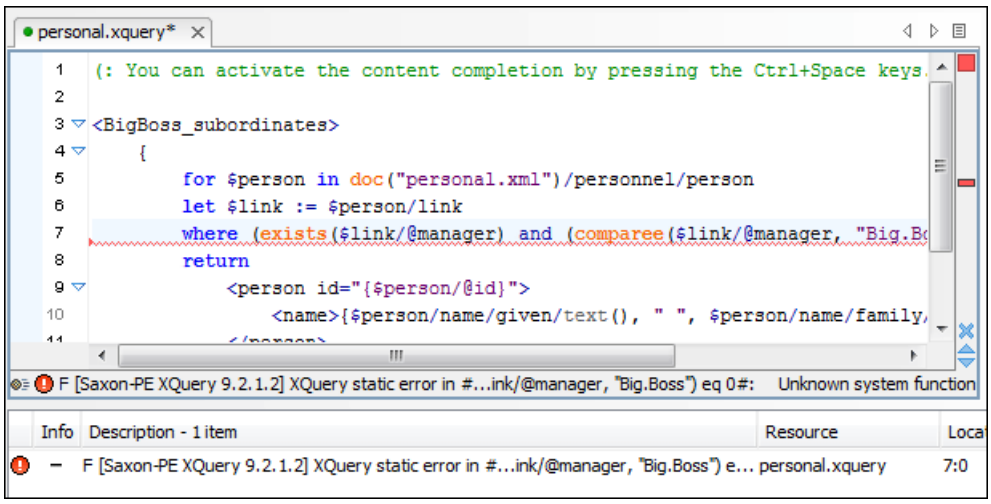


Figure 233: XQuery Validation

Please note that if you choose a processor that doesn't support XQuery validation you will receive a warning when trying to validate.

The **Validate** toolbar provides a button **Validation options** for quick access to the *XQuery options* in the Oxygen user preferences.

Other XQuery Editing Actions

The XQuery editor type offers a reduced version of *the popup menu available in the XML editor type*, that means:

- *the split actions,*
- *the folding actions*
- *the edit actions*
- a part of *the source actions:*
  - **To lower case**
  - **To upper case**
  - **Capitalize lines**
- open actions:
  - **Open file at Caret**
  - **Open file at Caret in System Application**

Transforming XML Documents Using XQuery

XQueries are very similar to the XSL stylesheets in the sense they both are 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* at the same time with the XSLT scenarios and can be managed in *the dialog **Configure Transformation Scenario*** or in *the **Scenarios** view*. The transformation performed can be based on the XML document specified in the **XML URL** field, or, if this field is empty, the documents referred from the query expression are used instead. 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 the processor Saxon 9.3.0.5 HE, Saxon 9.3.0.5 PE, Saxon 9.3.0.5 EE or 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 9.3.0.5 EE processor supports also XQuery 1.1 transformations. If *the option* ***Enable XQuery 1.1 support*** is enabled Saxon EE runs an XQuery transformation as an XQuery 1.1 one.

XQJ Transformers

This section describes the procedures necessary to apply 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.

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 **XQuery API for Java(XQJ)** in the **Type** combo box.
5. 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 will detect any implementation of `javax.xml.xquery.XQDataSource` and present it in **Driver class** field.
6. Select the most suited driver in the **Driver class** combo box.
7. 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. For avoiding the long time necessary for fetching the full result the **Evaluate as sequence** option of the XQuery transformation scenario should be used. This option fetches only the first chunk of the result and the user decides if he wants to fetch the next chunk after looking at the first chunk in the **Sequence** result view. The size of a chunk can be *set with a user option*.

The **Evaluate as sequence** option of the XQuery scenario must be selected in the **Output** tab of the dialog for editing the transformation scenario.

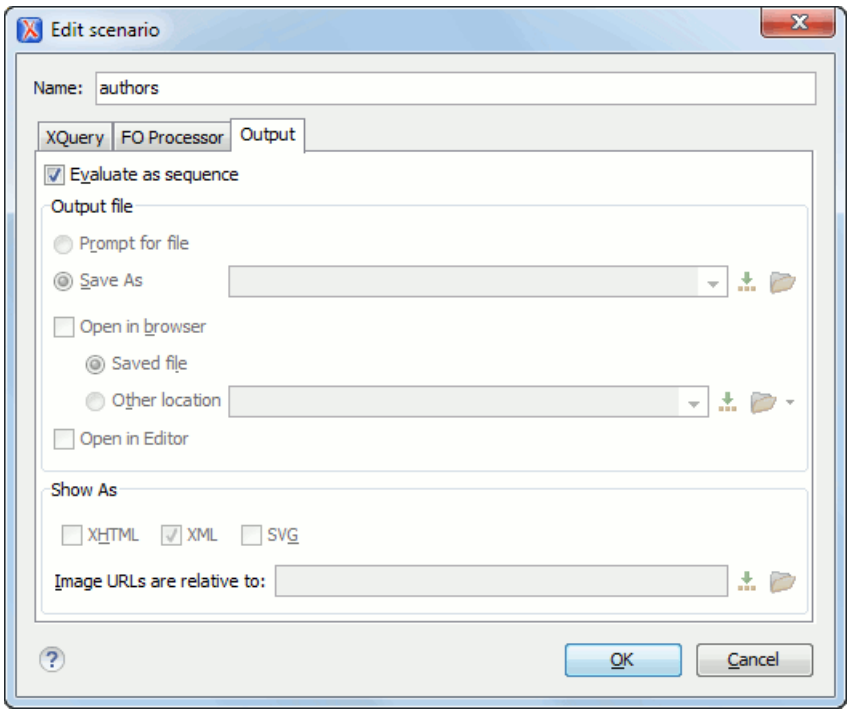


Figure 234: The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.

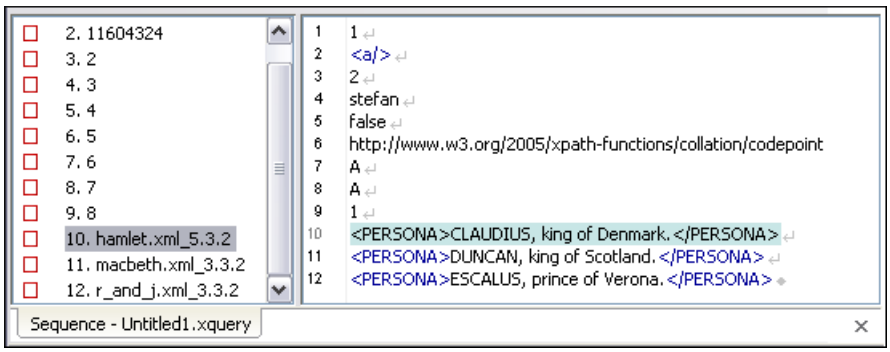


Figure 235: The XQuery transformation result displayed in Sequence view

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** - If checked, the specified Saxon configuration file will be used to specify the Saxon advanced options.
- **Recoverable errors** - Policy for handling recoverable errors in the stylesheet. Allows the user to choose how dynamic errors will be 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 three options:
  - **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** - 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** - Strips no whitespace before further processing. (However, whitespace will still be stripped if this is specified in the stylesheet using `xsl:strip-space`).
- **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.
- **Disable calls on extension functions** - If checked, calling external Java functions is disallowed.
- **Validation of the source file** - Available only for Saxon EE. It can have the values:
  - **Schema validation** - This mode requires an XML Schema and determines whether source documents should be parsed with schema-validation enabled.
  - **Lax schema validation** - This mode determines whether source documents should be parsed with schema-validation enabled if an XML Schema is provided.
  - **Disable schema validation** - This determines whether source documents should be parsed 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.
- **Enable XQuery 1.1 support** - If it is checked Saxon EE runs the XQuery transformation with the XQuery 1.1 support.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, a backup version is generated for any XML files that is updated with XQuery Update.

Updating XML Documents using XQuery

Using the bundled Saxon 9.3.0.5 EE XQuery processor Oxygen now 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.

Just choose Saxon 9.3.0.5 EE as a transformer in the scenario associated with XQuery files containing update statements and Oxygen 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 transformations and XQuery ones.

Overview

The **XSLT Debugger** and **XQuery Debugger** perspectives enable you to test and debug XSLT 1.0 / 2.0 stylesheets and XQuery 1.0 documents including complex XPath 2.0 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 for each step. At the same time, special views in the interface provide various types of debugging information and events useful for understanding the transformation process.

The user benefits of a rich set of features for testing and solving XSLT/XQuery problems:

- Support for XSLT 1.0 stylesheets (through the Saxon 6.5.5 and Xalan XSLT engines), XSLT 2.0 stylesheets and XPath 2.0 expressions that are included in the stylesheets (through the Saxon 9.3.0.5 XSLT engine) and XQuery 1.0 (through the Saxon 9.3.0.5 XQuery engine).
- Stepping capabilities: step in, step over, step out, run, run to cursor, run to end, pause, stop.
- Output to source mapping between every line of output and 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 debugger interface looks like below. It is comprised of 4 panes as follows:

- **Source document view (XML)** - Displays and allows editing of data or document oriented XML files (documents).
- **XSLT/XQuery document view (XSLT/XQuery)** - Displays and allows editing of XSL files(stylesheets) or XQuery documents.
- **Output document view** - Displays the transformed output that results from the input of a selected document (XML) and selected stylesheet (XSL) or XQuery document to the transformer. The result of transformation is dynamically written as the transformation is processed.There are two views for the output: a text view (with XML syntax highlight) and an XHTML view. For large output the XHTML view can be can disabled (see *Debugger Settings*).
- **Control view** - The control view provides functionality for configuration and control of debugging operations. It also provides a series of *Information views* types. This pane is comprised of two parts:
  - *Control toolbar*
  - *Information views*

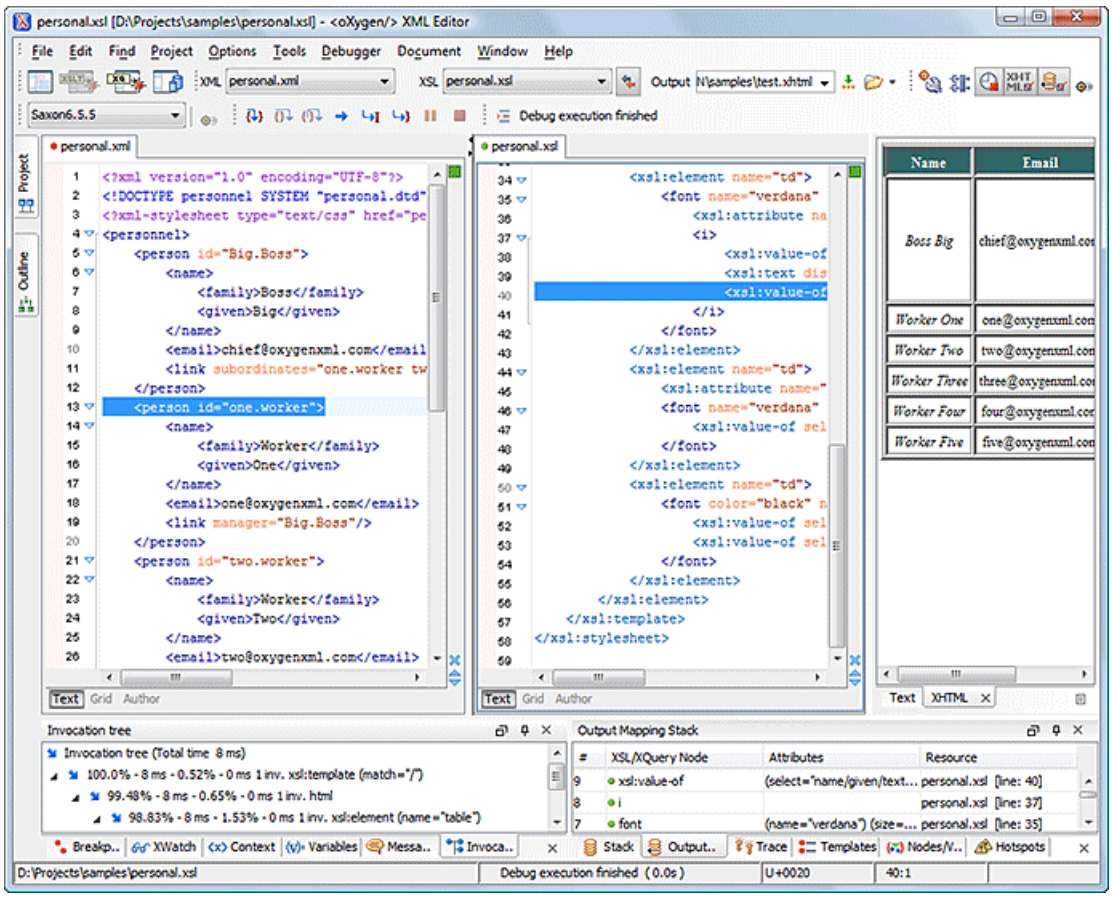


Figure 236: Debugger Mode Interface

XML documents and XSL stylesheets or XQuery documents that were opened in Editor perspective are automatically sorted into the first two panes. When multiple files of each type are opened, the individual documents and stylesheets are separated using the familiar tab management system of the Editor perspective. Selecting a tab brings the document or stylesheet into focus and enables editing without toggling back to the Editor perspective.

When editing in the Editor perspective the editor toolbar is displayed. In Debugger mode this toolbar is not available, however the functions are still accessible from *the **Document** menu* and *the context menus* that are activated by a right click of the mouse. On Windows the context menu can be displayed with the mouse on a right click or with the keyboard by pressing the special context menu key available on Windows keyboards. Bookmarks are replaced by breakpoints in Debugger perspective.

During debugging the current execution node is highlighted in both document (XML) and XSLT/XQuery views.

Control Toolbar

The toolbar contains all actions you need to configure and control the debug process. Items are described below from left to right as they appear in the toolbar.

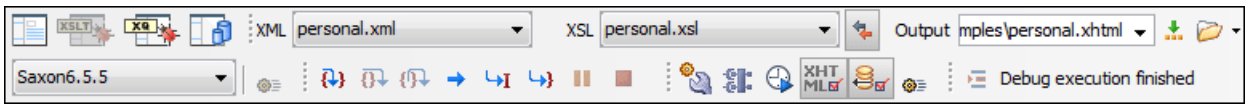


Figure 237: Control Toolbar

- **XML source selector** - The current selection represents the source document used by the transformation engine as input. A drop-down list contains all open files (the XML ones being emphasized). This gives you the possibility to use other file types as source. In a XQuery debugging session this selection field can be set to default value `NONE`, as usually XQuery documents do not require an input source.

- **XSL / XQuery selector** - The current selection represents the stylesheet or XQuery document to be used by the transformation engine. The selection list is filled-in with all opened files (the XSLT / XQuery ones 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. Enabled 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.
- **Edit extensions** - Add and remove the Java classes and jars used as XSLT extensions.
- **Turn on profiling** - Enables / Disables current transformation profiling.
- **Enable XHTML output** - Enables rendering of output to the XHTML output view during the transformation process. For performance issues, disable XHTML output when working with very large files. Also, the XHTML area is only able to render XHTML documents. In order to view the output result of other formats, such as HTML, save the Text output area to a file and use the required 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 instruction element / source context that generated it.
- **Debugger Preferences** - Quick link to *Debugger preferences page*.
- **XSLT / XQuery engine selector** - Lists the *available processors for debugging XSLT and XQuery transformations*.
- **XSLT / XQuery engine advanced options** - Advanced options available for Saxon 9.
- **Step into** - Starts the debugging process and runs until the next stylesheet node or the next XPath 2.0 expression step (next step in transformation).
- **Step over** - Executes the current stylesheet node (including its sub-elements) and goes to next node in document order (usually the next sibling of the current node) or to next step of an XPath 2.0 expression.

Figure 238: Step over

- **Step out** - Steps out to the parent node (equivalent to the *Step over* on the parent).

Figure 239: Step out

- **Run** - Starts the debugging process. The execution of the process is paused when a breakpoint is encountered. (see *breakpoints*).
- **Run to cursor** - Starts the debugging process and runs until one of the following conditions occur: the line of cursor is reached, a valid breakpoint is reached or end of execution.
- **Run to end** - Runs the transformation until the end, without taking into account enabled breakpoints (if any).
- **Pause** - Interrupts the current transformation. This 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** - 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.

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 session and is a source node corresponding to the XSL expression being 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.

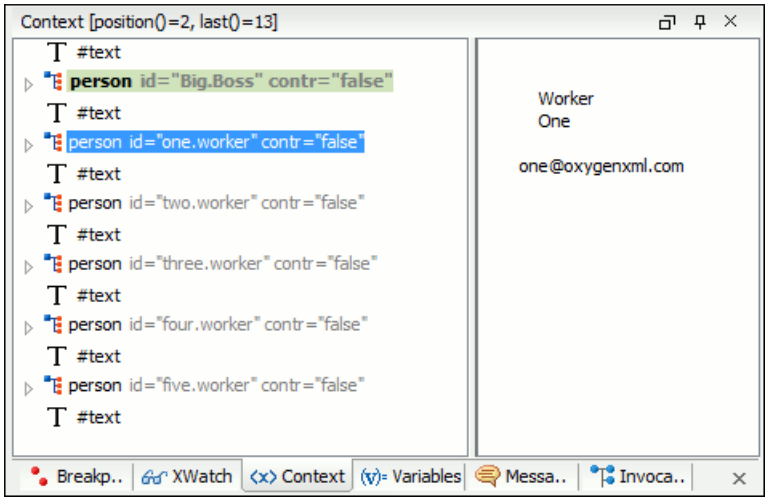


Figure 240: The Context node view

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 title bar displays the current element index and the number of elements that compose the current context (this information is not available if you choose Saxon 6 as processing engine).

XPath Watch (XWatch) View

This view shows XPath expressions to be evaluated during debugging. Expressions are evaluated dynamically as the processor changes its source context.

When the XPath expression is typed in the **Expression** column the usual content completion assistant is activated and *supports the process of composing the expression just like in the XSLT editor*.

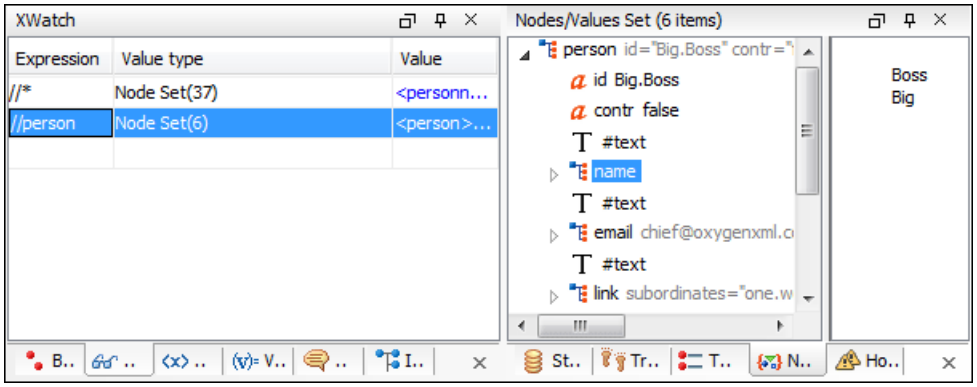


Figure 241: The XPath watch view

Table 3: XWatch columns

Column	Description
Expression	XPath expression to be evaluated (should be XPath 1.0 or 2.0 compliant).
Value	Result of XPath expression evaluation. Value has a type (see *the possible values* in the section *Variables View* on page 429). For Node Set 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. In case of an XPath error, you get an Error line.
- The expression list is not deleted at the end of transformation (it is preserved between debugging sessions).
- To insert a new expression click the last line on the expression column and enter it. As alternative right click and select the **Add** action. Press **(Enter)** on the cell to add and evaluate.
- To delete an expression click on its **Expression** column and delete its content. As alternative right click and select the **Remove** action. Press **(Enter)** on the cell to commit changes.
- If the expression result type is a Node Set you can click on it (**Value** column) and you will see on the right side its value. (see ***Nodeset** view*).
- The **Copy**, **Add**, **Remove** and **Remove All** actions are offered in every row's contextual menu.

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**.

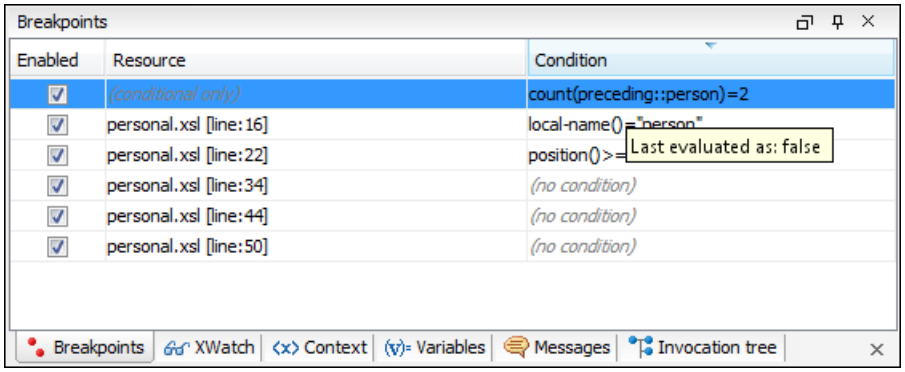


Figure 242: The Breakpoints View

Table 4: 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. The Entire path of resource file is available as tooltip.
Condition	XSLT/XQuery expression to be evaluated during debugging. The expression will be evaluated at every debug step.

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.

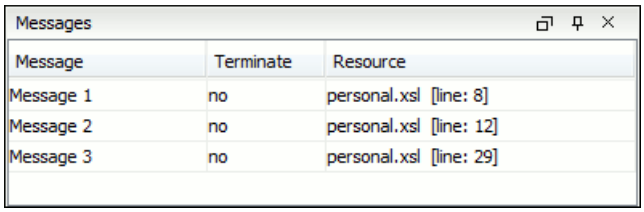


Figure 243: The Messages View

Table 5: 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 `xsl:message` instruction is defined and the message line number. The complete path of the resource is available as tooltip.

The following actions are available in the contextual menu:

- **Go to** - Highlight the XSL fragment that generated the message.
- **Copy Value** - 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 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.

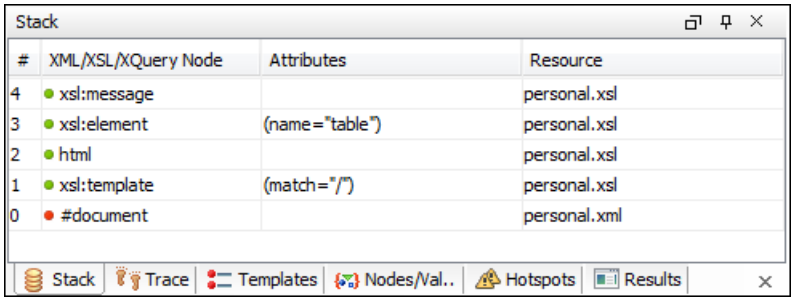


Figure 244: The Stack View

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 6: 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 `id="value"` pairs).
Resource	Resource file where the node is located. The entire path is available as tooltip.

Important: Remarks:

- Clicking a record from the stack highlights that node's location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: `xsl:template` using Saxon and `template` using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

Output Mapping Stack View

This view is useful at the end of the transformation and shows the whole stack of XSLT templates/XQuery elements that generated a specific area of the output. It provides *context data in addition to the highlight of the XML element and the XSLT template/XQuery element* available in the editor panel and in the **Stack** view and **Trace** view.

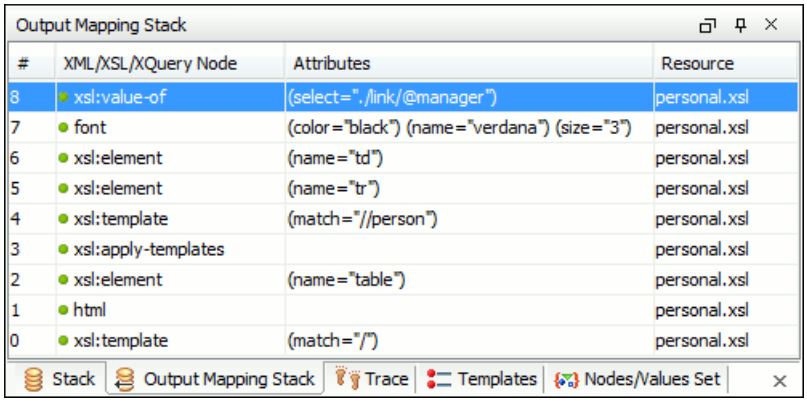


Figure 245: The Output Mapping Stack view

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 7: 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.

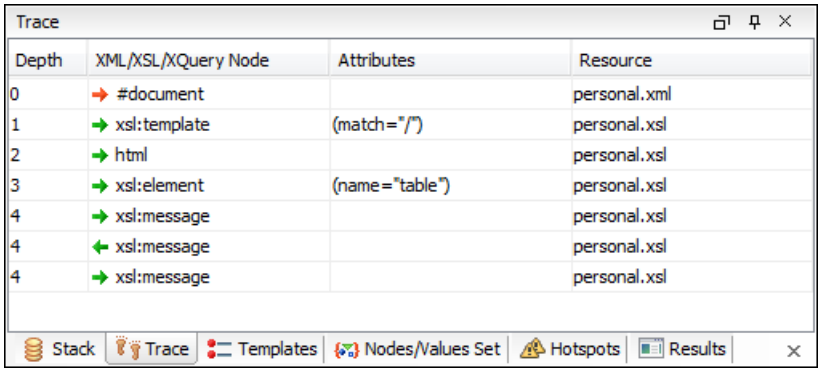


Figure 246: The Trace History View

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 8: Trace History columns

Column	Description
Depth	Shows you how deep the node is nested in the XML or stylesheet structure. The bigger the number, the more nested the node is. A depth 0 node is the document root.
XML/XSLT/XQuery Node	Represents the node from the processed source or stylesheet document. One particular node is the document root, noted as `#document`. Every node is preceded by an arrow that represents what action was performed on it (entering or leaving the node).
Attributes	Attributes of the node (a list of `id="value"` pairs).
Resource	Resource file where the node is located. The complete path of the resource file is provided as tooltip.

Important: Remarks:

- Clicking a record highlights that node's location inside the resource.
- Only the Saxon processor shows the element attributes.
- The Xalan processor shows also the built-in rules.

Templates View

The `xsl:template` is the basic element for stylesheets transformation. This view is only available during XSLT debugging sessions and shows all `xsl:template` instructions used by the transformation. By seeing the number of hits for each of the templates you get an idea of the stylesheet coverage by template rules with respect to the input source.

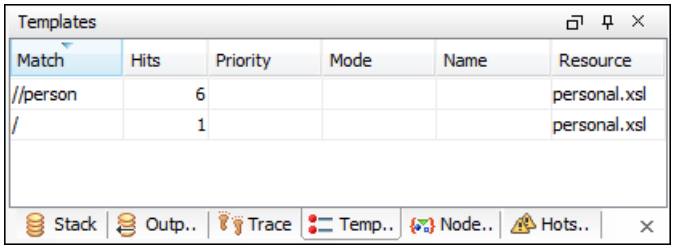


Figure 247: The Templates view

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 9: Templates columns

Column	Description
Match	The `match` attribute of the `xsl:template`.
Hits	The number of hits for the `xsl:template`. Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The `mode` attribute of the `xsl:template`.
Name	The `name` attribute of the `xsl:template`.
Resource	The resource file where the template is located. The complete path of the resource file is available as tooltip.

Important: Remarks:

- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

Node Set View

This view is always used in relation with *The **Variables** view* and *the **XWatch** view*l. 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.

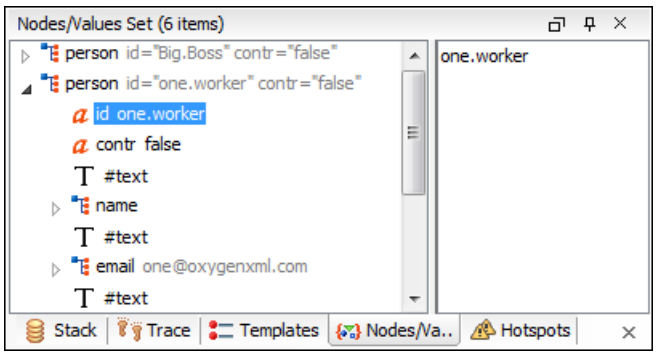


Figure 248: The Node Set view

The nodes / values set is presented in a tree-like fashion. The total number of items is presented in the title bar. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel.

Important: Remarks:

- In case of longer values in the right side panel the interface shows three suspension points (...) at the end. A more detailed value is available as tooltip.
- Clicking a record highlights the location of that node into the source or stylesheet view.

Variables View

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen 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 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.

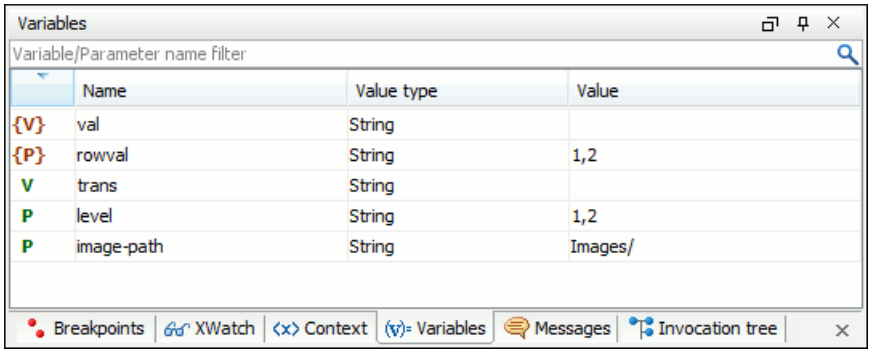


Figure 249: The Variables View

Table 10: 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

For XSLT 2.0 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.

Steps in a Typical Debug Process

To debug a stylesheet or XQuery document follow the procedure:

1. *Open the source XML document* and *the XSLT/XQuery document.*
2. If you are in the Editor perspective switch to the XSLT Debugger perspective or XQuery Debugger perspective with one of the actions (here explained for XSLT):
   - Menu **Window** > **Open perspective** > **XSLT Debugger** or the toolbar button **XSLT Debugger**
   - Menu **Document** > **XML Document** > **Debug Scenario** or the toolbar button **Debug Scenario**. This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.
3. Select the source XML document in the XML source selector of *the Control toolbar*. In case of XQuery debugging if your XQuery document has no implicit source set the source selector value to **NONE**.
4. Select the XSLT/XQuery document in the XSLT/XQuery selector of *the Control toolbar*.
5. Set XSLT/XQuery parameters from the button available on *the Control toolbar*.
6. *Set one or more breakpoints*.
7. Step through the stylesheet using the buttons available on *the Control toolbar*:
   - **Step into**
   - **Step over**
   - **Step out**
   - **Run**
   - **Run to cursor**
   - **Run to end**
   - **Pause**
   - **Stop**
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 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. Place your cursor on the line where you want the breakpoint to be in the XML source document or the XSLT / XQuery document.

You can set breakpoints on XML source only for XSLT debugging sessions.

2. Select **Edit** > **Breakpoints** > **Breakpoints quick creation** or directly 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.

Output to source mapping is a powerful feature that makes this output to source mapping persistent. 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 XSLT Debugger perspective or the XQuery Debugger perspective with one of the actions (here explained for XSLT):
   - Go to menu **Window** > **Open perspective** > **XSLT Debugger** or the toolbar button  **XSLT Debugger**
   - Go to menu **Document** > **XML Document** > **Debug scenario** or the toolbar button  **Debug scenario** . This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.
2. Select the source XML document in the XML source selector of *the Control toolbar*. In case of XQuery debugging without an implicit source choose the NONE value.
3. Select the XSLT / XQuery document in the XSLT / XQuery selector of *the Control toolbar*.
4. Select the XSLT / XQuery engine in the XSLT / XQuery engine selector of *the Control toolbar*.
5. Set XSLT / XQuery parameters from the button available on *the Control toolbar*.
6. Apply the XSLT stylesheet or XQuery transformation using the button  **Run to end** available on *the Control toolbar*.
7. Inspect the mapping by clicking a section of the output from the **Text** view tab or from the **XHTML** view tab of the *Output document view* .

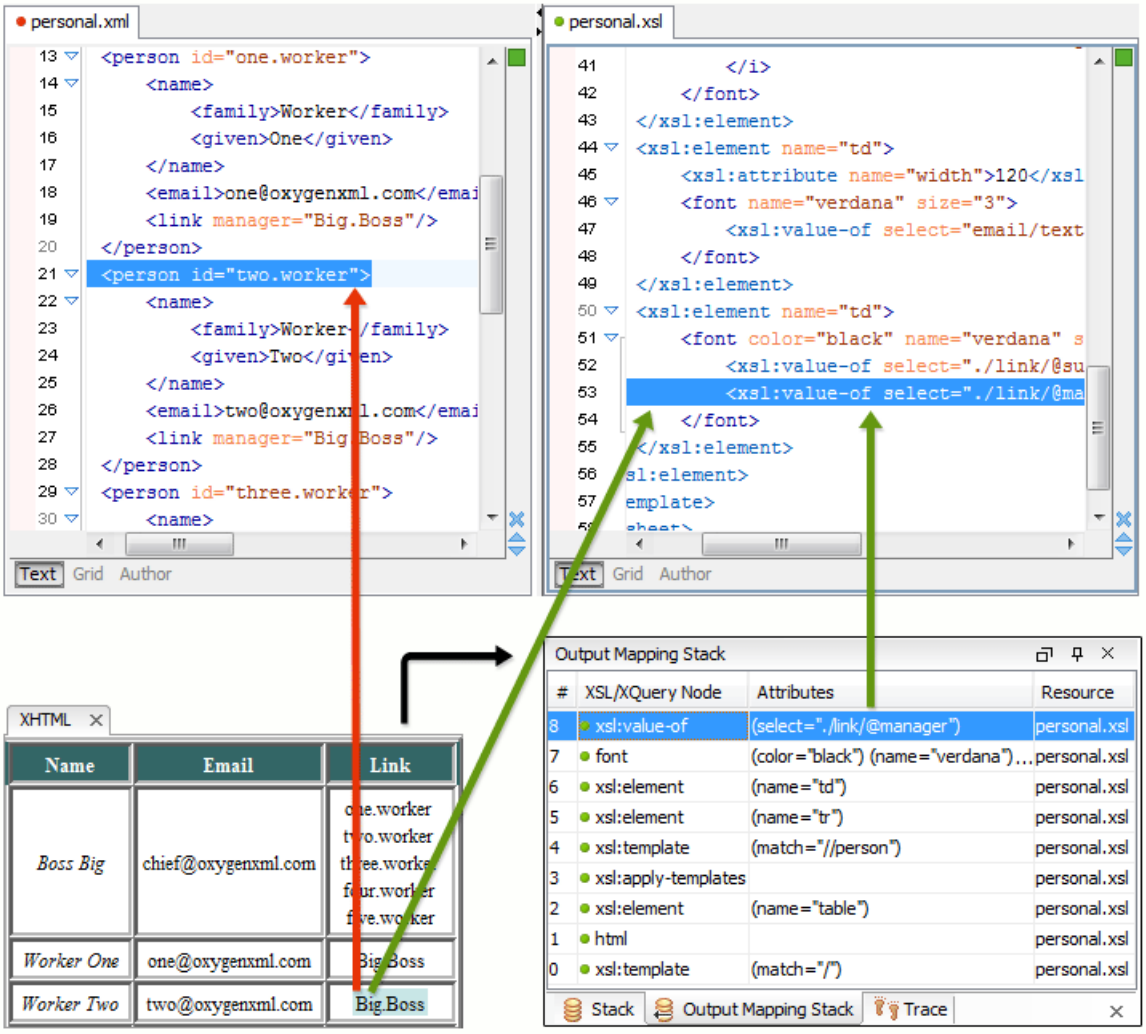


Figure 250: XHTML Output to Source Mapping

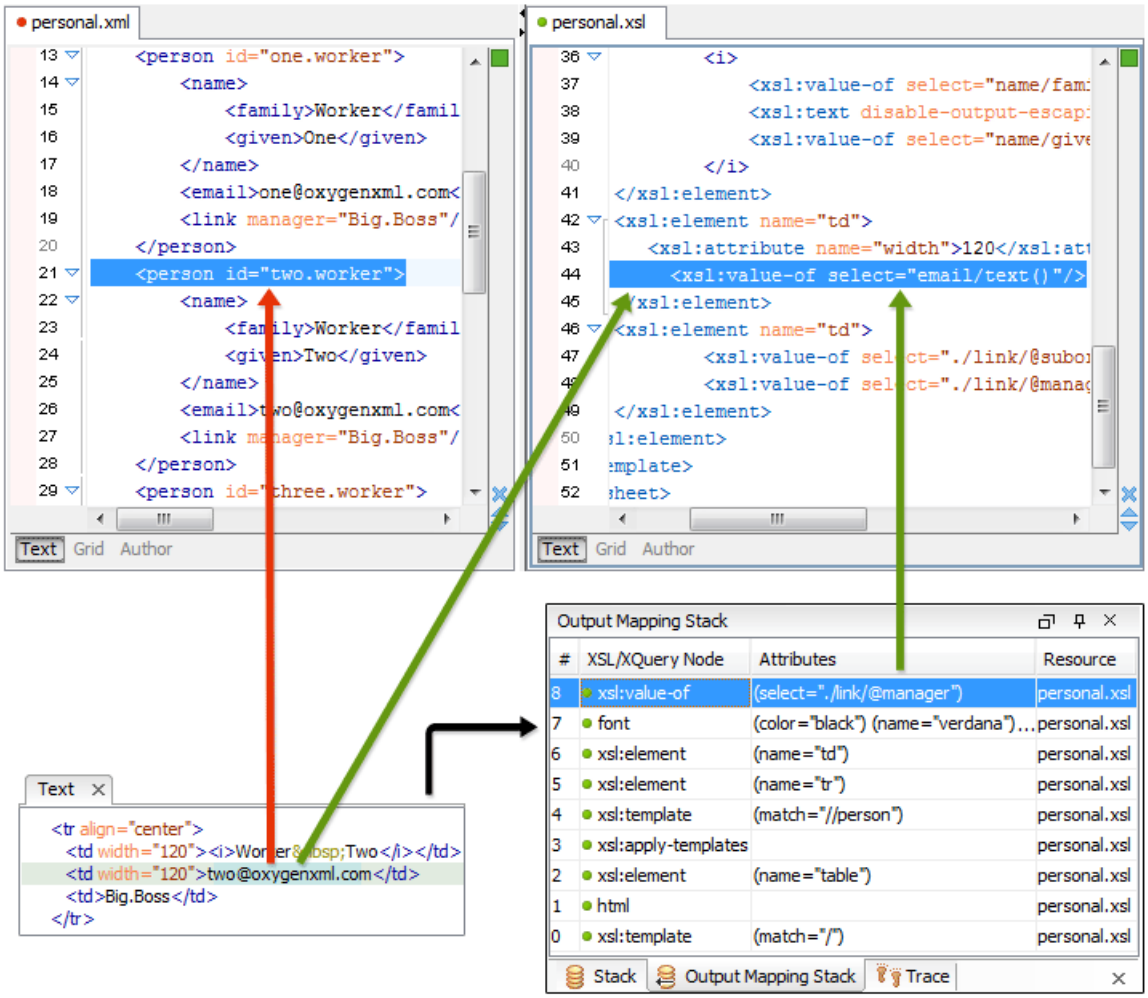


Figure 251: Text Output to Source Mapping

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. For stepping into Java classes, inspecting variable values and setting breakpoints in Java methods you should set up a Java debug configuration in an IDE like the Eclipse SDK as described below.

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, etc.).

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 9.3.0.5 HE (Home Edition) - a limited version of the Saxon 9 processor, capable of running XSLT 1.0, XSLT 2.0 basic and XQuery 1.0 transformations, available in both the XSLT debugger and the XQuery one,
- Saxon 9.3.0.5 PE (Professional Edition) - capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery one,
- Saxon 9.3.0.5 EE (Enterprise Edition) - a schema aware processor, capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones, XSLT 2.0 schema aware ones and XQuery 1.0 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

Profiling XSLT Stylesheets and XQuery Documents

Topics:

- *Overview*
- *Viewing Profiling Information*
- *Working with XSLT/XQuery Profiler*

This chapter explains the user interface and how to use the profiler for finding performance problems in XSLT transformations and XQuery ones.

Overview

Whether you are trying to identify a performance issue that is causing your production XSLT/XQuery transformation to not meet customer expectations or you are trying to proactively identify issues prior to deploying your XSLT/XQuery transformation, using the XSLT/XQuery profiler feature is essential to helping you save time and ultimately ensure a better performing, more scalable XSLT/XQuery transformation.

The XSLT/XQuery profiling feature can use any available XSLT/XQuery processors that could be used for debugging and it is available from the debugging perspective.

Enabling and disabling the profiler is controlled by the *Profiler button* from the *debugger control toolbar*. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you should set it before starting the transformation.

Viewing Profiling Information

This section explains the views that display the profiling data collected by the profiles during the transformation.

Invocation Tree View

This view shows a top-down call tree representing how XSLT instructions or XQuery expressions are processed.

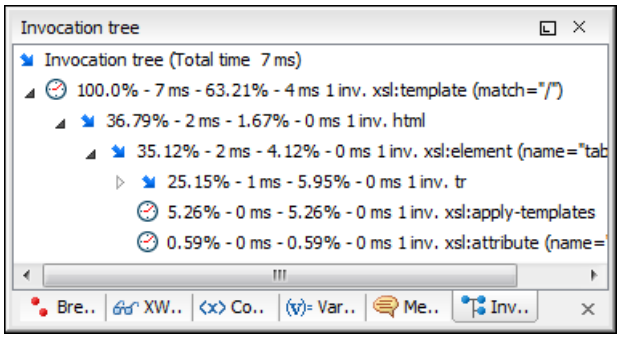


Figure 252: Invocation tree view

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 call tree time.
- - Points to a call whose inherent time is significant compared to its call tree time (greater than 1/3rd of its call tree 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 path.

- An instruction name which contains also the attributes description.

Hotspots View

This view shows a list of all instruction calls which lie above the threshold defined in the ***XSLT/XQuery profiler settings***
.

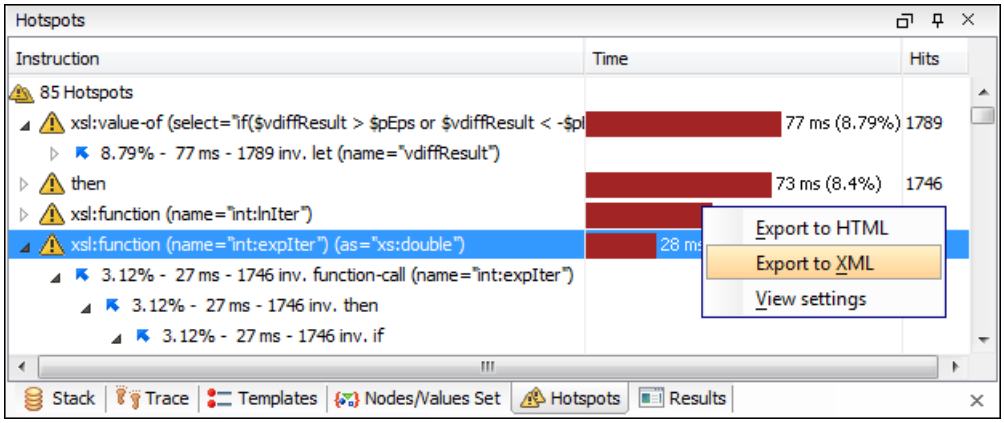


Figure 253: Hotspots View

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 path.
- An invocation count which shows how often the hotspot has been invoked on this 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.

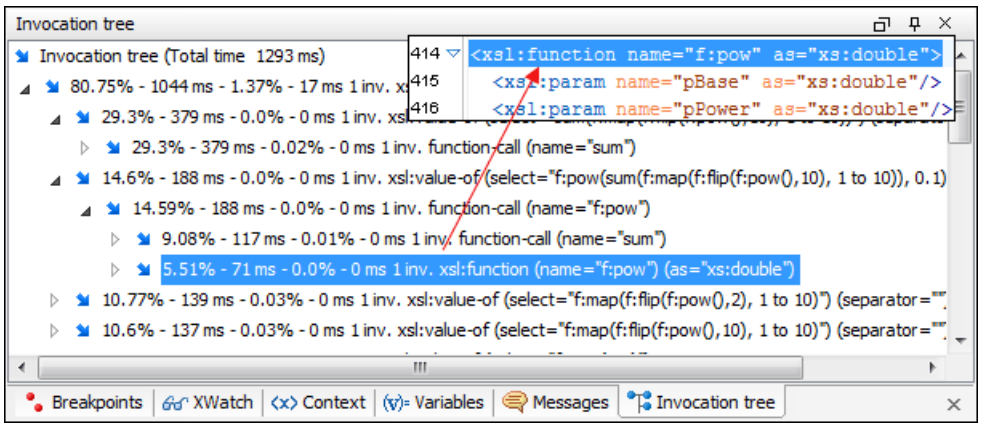


Figure 254: Source backmapping

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 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 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 distribution (see the subfolder `frameworks/profiler/` of the Oxygen 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.

Chapter 14

Comparing and Merging Documents

Topics:

- *Directories Comparison*
- *Files Comparison*
- *XML Diff API*

In large teams composed either of developers or technical writers, the usage of a shared repository for the source or document files is a must. Often many authors are changing the same file at the same time.

Finding what has been modified in your files and folders can be hard. If your data is changing, you can benefit from accurate identification and processing of changes in your files and folders with Oxygen XML Editor 's features for comparing files and directories. These are powerful and easy to use tools that will do the job fast and thoroughly. With the new possibilities of differencing and merging, it is now easy to manage multiple changes.

Oxygen XML Editor provides a simple means of performing file and folder comparisons. You can see the differences in your files and folders and also you can merge the changes.

There are two levels on which the comparison can be done, namely comparing directories or comparing individual files. These two operations are available from the Tools menu.

Also the comparison tool can be started using command line arguments. In the installation folder there are 2 executable shells (`diffFiles.bat` and `diffDirs.bat` on Windows, `diffFiles.sh` and `diffDirs.sh` on Unix/Linux, `diffFilesMac.sh` and `diffDirsMac.sh` on Mac OS X). You can give one or two command line arguments to each of these shells.

For example, to start the comparison between 2 directories on Windows use:

```
diffDirs.bat "c:\Program Files" "c:\ant"
```

Note that if there are spaces in the path names, the paths need to be surrounded by quotes. Also one argument can be missing in which case the second directory will be chosen manually by the user.

The same goes for the files diff utility as well.

If you run the diff tool from the command line (`diffFiles.exe` or `diffFiles.bat` on Windows, `diffFiles.sh` on Linux, `diffFilesMac.sh` on Mac OS X), you must specify one or two parameters, because Diff Files perform only two-way comparing.

Directories Comparison

The directories comparison result is presented as a tree of files and directories. The directories that contain different files are expanded automatically, so you can focus directly on the differences. You can merge the directories contents using the copy actions. A double click or an Enter key on a line with a pair of files starts *comparing the file content* of the two files from that line in the **Compare Files** window. Please note that the content is compared only in case of known file types, that is the files associated with the built-in editors and the file types associated with a built-in editor when the user was prompted to specify such an association (when opening for the first time a file of an unknown type).

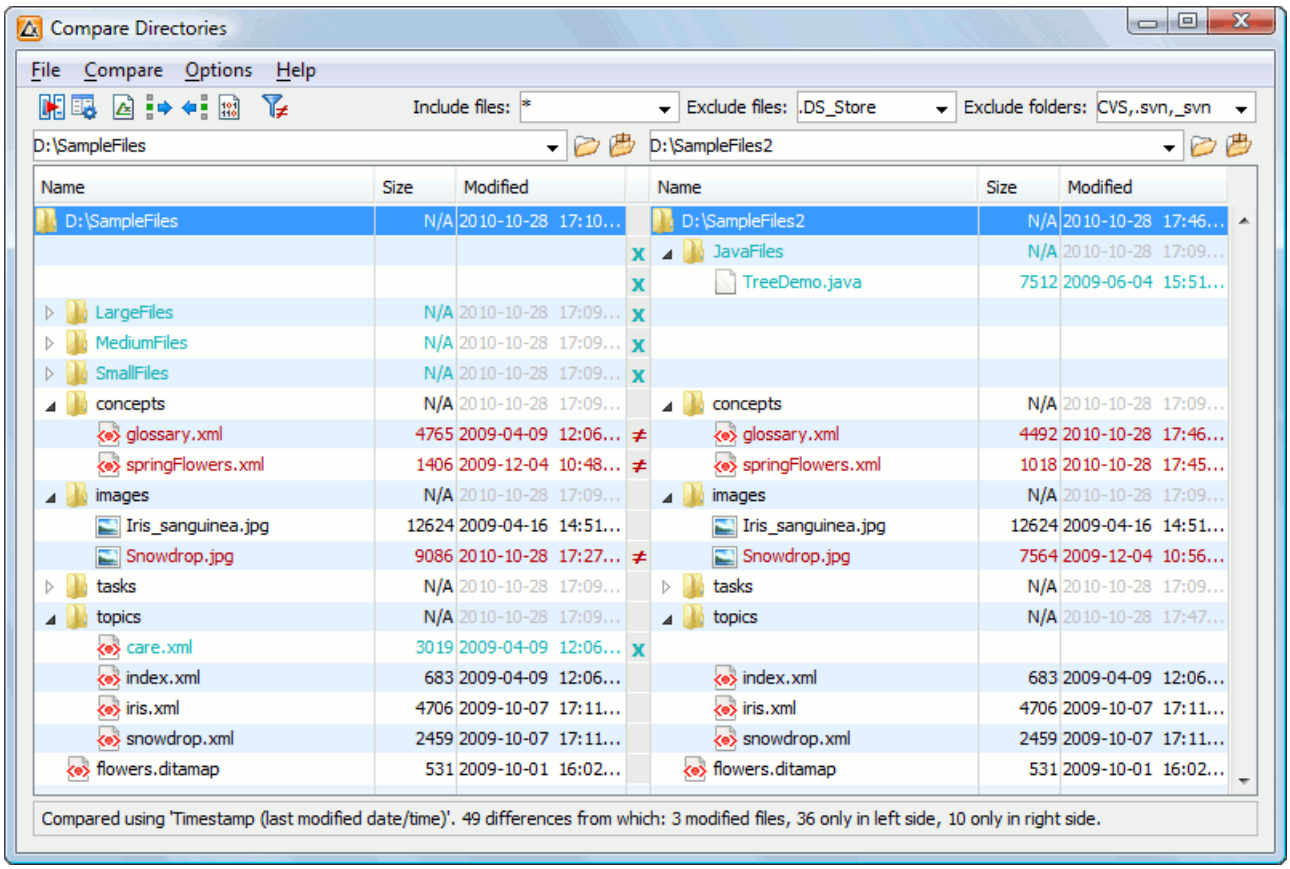


Figure 255: The Compare Directories Window

Directories Comparison User Interface

This section explains the user interface of the **Directories Comparison** window.

Compare Menu

This menu contains the following action:

- **Perform directories differencing** - Looks for differences between the two directories displayed in the left and right side of the application window.
- **Perform files differencing** - Compares the currently selected files.
- **Copy change from left to right** - Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).
- **Copy change from right to left** - Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).

Compare Toolbar

The toolbar contains the following actions:

Figure 256: The Compare toolbar

- **Perform directories differencing** - Looks for differences between the two directories displayed in the left and right side of the application window.
- **Diff Options** - Opens the Directories Comparison preferences page.
- **Perform files differencing** - Compares the currently selected files.
- **Copy change from left to right** - Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).
- **Copy change from right to left** - Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).
- **Binary Compare** - Performs a byte-level comparison on the selected files.
- **Show Only Modifications** - Displays a more uncluttered file structure by hiding all identical files.
- Files and folders filters - Differences can be filtered using three filter boxes: **Include files, Exclude files, Exclude folders**. They come with predefined values and are editable to allow more custom values. All of them accept multiple comma separated values and the ***** and **?** wildcards. For example, to filter out all jpeg and gif image files, edit the **Exclude files** filter box to read **`*.jpeg, *.png`**. Each filter keeps a list with the latest 15 filters applied in the drop-down list of the filter box.

Directories Selector

To open the directories you want to compare, select a folder from each **Browse for local directory** button. Oxygen XML Editor keeps track of the folders you are currently working with and those you opened in this window. You can see and select them from the two combo-boxes.

If you want to compare two archives' content you can select the archives from the **Browse for archive file** button.

👉 **Tip:** By default Oxygen XML Editor treats supported archives as directories and the comparison is also done with the files inside them. You can disable this behaviour by unchecking the **Look in archives** checkbox from the *Diff preferences page* .

Comparison Result

The directory comparison result is presented using two tree-like structures, showing files and folders names, size and modification date.

Figure 257: Comparison result

A column holding graphic symbols separate the two tree-like structures. The graphic symbols can be:

- an "X" sign, when a file or a folder exists only in one of the compared directories;
- a "not-equal" sign, if a file exists in both directories, but the content is different. The same sign appears when a collapsed folder contains modified files.

The color used for painting the sign and the directory or file name can be customized in the *Directories Comparison / Appearance* preferences page.Oxygen XML Editor You can double-click the line marked with the "not-equal" sign and a new **File Content Comparison** window is opened, showing the differences between the two files.

Compare Images

By double-click a line containing two different images a compare images dialog is displayed. The dialog presents the images in the left and right part scaled to fit the view available area. You can use the contextual menu actions to scale the images at their original size or scale them down to fit in the view available area.

The supported image types are: GIF, JPG / JPEG, PNG, BMP.

Files Comparison

The comparison of a pair of files is done by opening them in two editors arranged in a side-by-side layout. The line numbers on the left side of each editor help you to identify quickly the locations of the differences.

You can edit both the source and the target file. The differences are refreshed when you save the modified document.

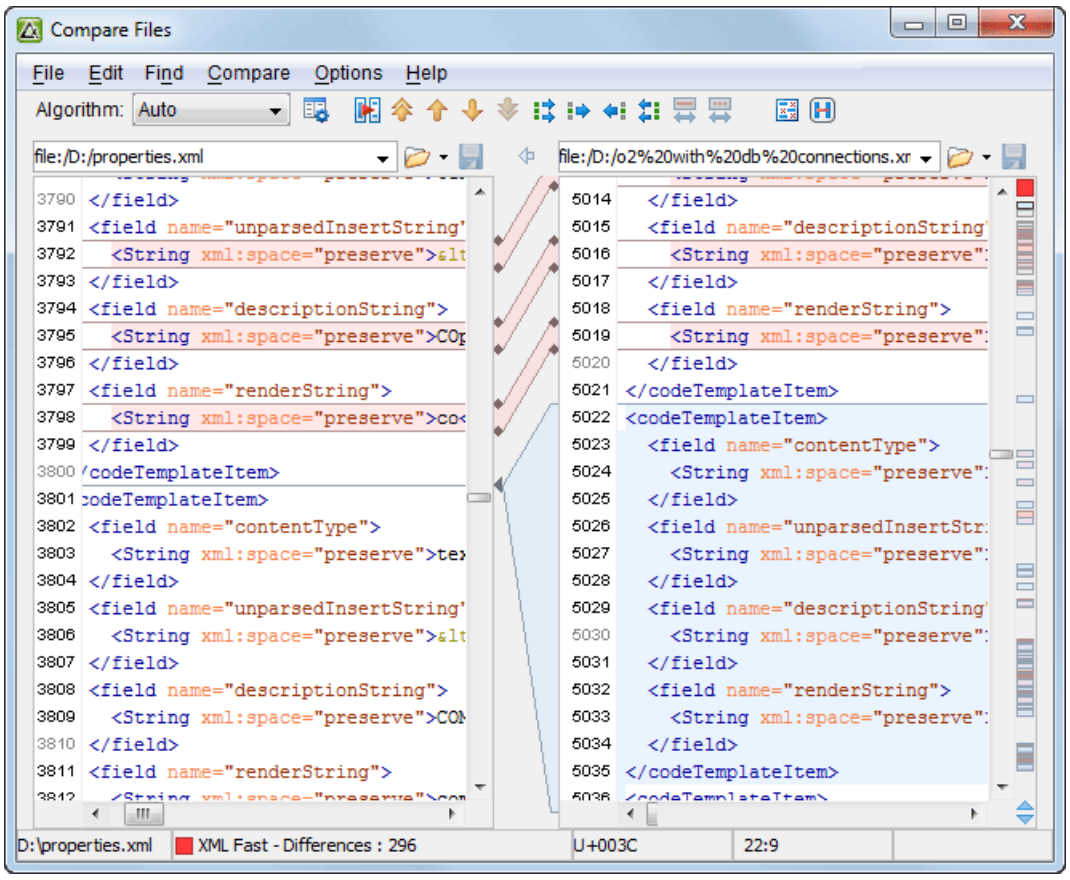


Figure 258: The Compare Files Window

Main Menu

This section explains the menu actions of the **Files Comparison** window.

File Menu

The following actions are available:

- **Source** - The file is displayed in the left side of the application window
  - **Source** > **Open** - Browses for a source file.
  - **Source** > **Open URL** - Opens URL to be used as a source file. See *Open URL* for details.
  - **Source** > **Open File from Archive** - Browses an archive content for a source file.
  - **Source** > **Save** - Saves the changes made in the source file.
- **Target** - The file is displayed in the right side of the application window
  - **Target** > **Open** - Browses for a target file.
  - **Target** > **Open URL** - Opens URL to be used as a target file. See *Open URL* for details.
  - **Target** > **Open File from Archive** - Browses an archive content for a source file.
  - **Target** > **Save** - Saves the changes made in the target file.
- **Exit** - Quits the application.

Edit Menu

The following actions are available:

- **Cut** - Cut selection to clipboard from the local file currently opened in the Editor view or the Compare view.
- **Copy** - Copy selection to clipboard from the local file currently opened in the Editor view or the Compare view.
- **Paste** - Paste selection from clipboard in the local file currently opened in the Editor view or the Compare view.
- **Undo** - Undo edit changes in the local file currently opened in the Editor view or the Compare view.
- **Redo** - Redo edit changes in the local file currently opened in the Editor view or the Compare view.

Find Menu

The find actions are the following:

- **Find/Replace** - Perform find/replace operations in the local file currently opened in the **Editor** view or the **Compare** view.
- **Find Next** - Go to the next match using the same options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.
- **Find Previous** - Go to the previous match using the same options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.

Compare Menu

The following actions are available in this menu:

Perform Files Differencing - Performs a comparison between the source and target files.

Go to First Modification - Selects the first found difference. (The button becomes available if the selection is not on the first modification).

Go to Previous Modification - Navigates backwards in the list of found differences. (The button becomes available if the selection is not on the first modification). When the first modification in the current file is reached, the behaviour of this action is controlled by *Navigation* options.

Go to Next Modification - Navigates forwards in the list of found differences. (The button becomes available if the selection is not on the last modification). When the last modification in the current file is reached, the behaviour of this action is controlled by *Navigation* options.

Go to Last Modification - Selects the last found difference. (The button becomes available if the selection is not on the last modification)

Copy All Changes from Left to Right - Copies all changes from source to target file.

Copy All Changes from Right to Left - Copies all changes from target to source file.

Copy Change from Left to Right - Copies the selected difference from source to target file.

Copy Change from Right to Left - Copies the selected difference from target to source file.

Show Modification Details at Word Level - Provides a word-level comparison of the differences selected in the source and file.

Show Modification Details at Char Level - Provides a character-level comparison of the differences selected in the source and file.

Options Menu

- **Preferences** - Opens the options pages.

- **Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page. Here you can configure all keyboard shortcuts available for menu items available in Oxygen XML Diff.
- **Reset Global Options** - Resets options to their default values.
- **Import Global Options** - Allows you to import options you have previously exported.
- **Export Global Options** - Allows you to export the current options to a file.

Help Menu

- **Help** - Opens the Help dialog.
-

Compare Toolbar

This toolbar contains the operations that can be performed on the source and target files.

Figure 259: The Compare Toolbar

The following actions are available:

- **Algorithm** - This option box allows you to select one of the 6 available compare algorithms:
  - *Characters* algorithm computes the differences at character level;
  - *Words* algorithm computes the differences at word level;
  - *Lines* algorithm computes the differences at line level;
  - *Syntax Aware* for file types known by Oxygen XML Editor . It computes the differences considering the documents syntax.
  - *XML Fast* works on larger files but it is less precise than XML Accurate.
  - *XML Accurate* works best on small files XML files.
  - *Auto* selects the most appropriate algorithm, based on the files content and size. By default, an **Auto** mode is selected.
- **Diff Options** - Opens the *Files Comparison page*.
- **Perform Files Differencing** - Performs a comparison between the source and target files.
- **Go to First Modification** - Selects the first found difference. (The button becomes available if the selection is not on the first modification).
- **Go to Previous Modification** - Navigates backwards in the list of found differences. (The button becomes available if the selection is not on the first modification). When the first modification in the current file is reached, the behaviour of this action is controlled by *Navigation* options.
- **Go to Next Modification** - Navigates forwards in the list of found differences. (The button becomes available if the selection is not on the last modification). When the last modification in the current file is reached, the behaviour of this action is controlled by *Navigation* options.
- **Go to Last Modification** - Selects the last found difference. (The button becomes available if the selection is not on the last modification)
- **Copy All Changes from Left to Right** - Copies all changes from source to target file.
- **Copy Change from Left to Right** - Copies the selected difference from source to target file.
- **Copy Change from Right to Left** - Copies the selected difference from target to source file.

- **Copy All Changes from Right to Left** - Copies all changes from target to source file.
- **Show Modification Details at Word Level** - Provides a word-level comparison of the differences selected in the source and file.
- **Show Modification Details at Char Level** - Provides a character-level comparison of the differences selected in the source and file.
- **Ignore Whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.
- **Enable/Disable scrolling synchronization** - Synchronizes scrolling of the two open files, so that a selected difference can be seen on both sides of the application window. This action enables/disables the previous described behavior.
-

Files Selector

To open the source and target files where you want to see the differences, select a file from the **Open** or **Open URL** button. Oxygen XML Editor keeps track of the files you are currently working with and those you opened in this window. You can see and select them from the two combo-boxes.

You can also save the changes in the source file or the target file by clicking the corresponding "Save" button.

File Contents Panel

The files are opened in two side-by-side editors. The text view is used, offering a better view of the differences.

The two editors are kept in sync, so if you scroll the text in one of them, the other one will also scroll to show the difference. The differences are indicated using highlights connected through colored areas. You can use the **Go to modification** buttons to navigate between differences or simply select a change by clicking it in the overview ruler located in the right-most part of the window. Also, the overview ruler contains a success indicator in its upper part that will turn green in case the are no differences and red if differences are found. You can also do this by clicking a colored area between the text editors.

You can edit either the source or the target file. The differences are refreshed when you save the modified document.

Both editors provide a contextual menu that contains the **Cut**, **Copy**, **Paste** and **Select all** actions. The **Find/Replace** dialog is displayed by pressing **Ctrl+F** (**Cmd+F** on Mac). Also there are available the **Find/Replace** options: **F3** used to perform another search using the last search configuration, and **Shift+F3** to perform another search in backward direction using the last search configuration.

If the compared blocks of text are too large and you want to see the differences at a finer level, you can use the comparison at **Word** or **Character** level.

Word Level Comparison

This option is only available if modifications exist between the source and the target file. You can go to Word Level Comparison by clicking the **Show modification details at word level** button from the Compare Panel or from the Operations menu.

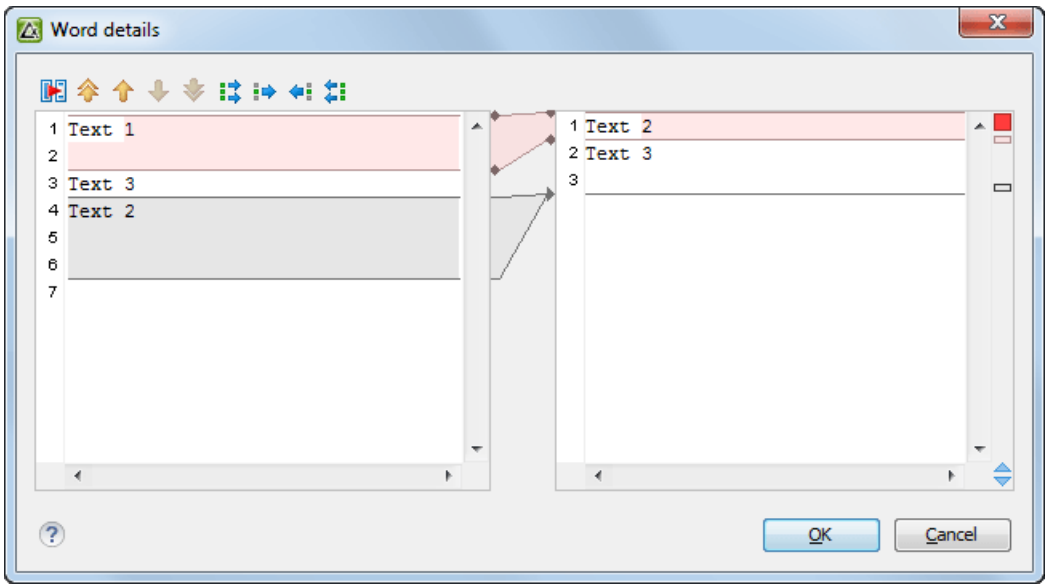


Figure 260: Word Level Comparison

Character Level Comparison

This option is only available if modifications exist between the source and the target file. You can go to Character Level Comparison by clicking the **Show Modification Details at Char Level** button from the compare panel or from the **Compare** menu.

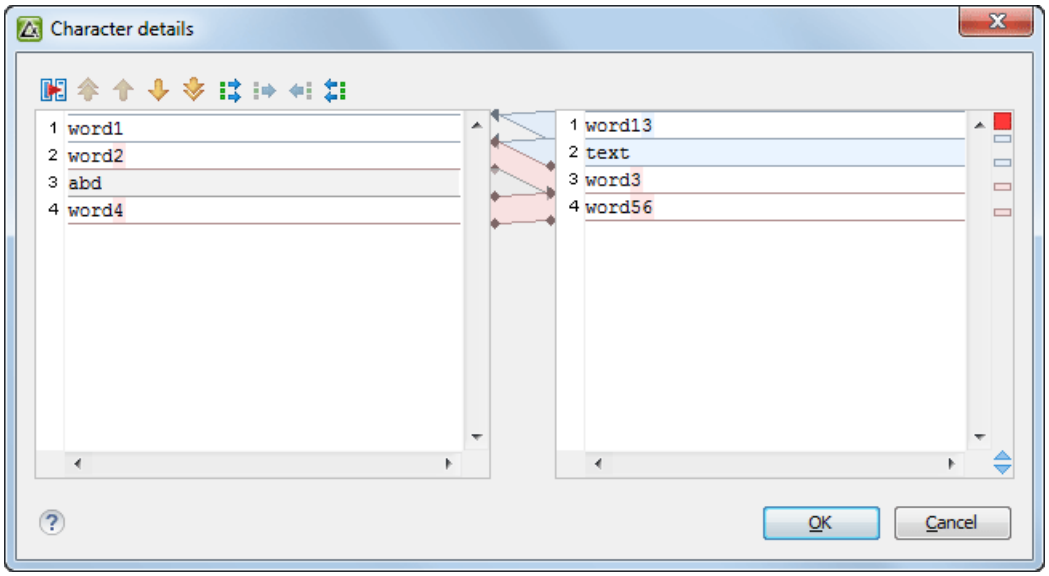


Figure 261: Character Level Comparison

XML Diff API

The following interface is available for calling the XML diff processor from a custom Java application:

- `ro.sync.diff.api.DifferencePerformer` - It compares two resources of a given content type using a set of options. It has the following methods:

- `performDiff` - Perform a diff operation between the two specified resources. It returns a list with the differences. The parameters are the following:
  1. leftContentReader - A value of type `java.io.Reader` that provides the content of the first resource.
  2. rightContentReader - A value of type `java.io.Reader` that provides the content of the second resource.
  3. leftSystemId - A string value that is the location of the first resource.
  4. rightSystemId - A string value that is the location of the second resource
  5. contentType - A constant from the `ro.sync.diff.api.DiffContentTypes` interface.
  6. diffOptions - The user options like algorithm strength, ignore whitespaces, ignore comments, merge adjacent differences, etc. It is a value of type `ro.sync.diff.api.DiffOptions`.
  7. diffProgressListener - An object that will be notified about the progress of the diff operation. It is a value of type `ro.sync.diff.api.DiffProgressListener`.
- `stop` - Signal to the diff performer that it must stop.

An example of this interface can be found in the class `ro.sync.diff.api.sample.DiffXMLFilesSample` which is included in the *XML Diff SDK*.

Chapter 15

Working with Archives

Topics:

- *Browsing and Modifying Archive Structure*
- *Working with EPUB*
- *Editing Files From Archives*

Oxygen offers the means to manipulate files directly from ZIP type archives. By manipulation one should understand opening and saving files directly in archives, browsing and modifying archive structures. The archive support is available for all ZIP-type archives, which includes:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

This means that you can modify, transform, validate files directly from *OOXML* or *ODF* packages. The structure and content of an EPUB book, OOXML file or ODF file *can be opened, edited and saved* as for any other ZIP archive.

You can transform, validate and perform many other operations on files directly from an archive. When selecting an URL for a specific operation like transformation or validation you can click the  **Browse for archived file** button to navigate and choose the file from a certain archive.

Browsing and Modifying Archive Structure

You can open an archive in the **Archives Browser** view doing one of the following:

- Open an archive from the ***Project** view*;
- Choose an archive in the Oxygen file chooser dialog;
- Drag an archive from the file explorer and drop it in the **Archives Browser** view.

When displaying an archive, the **Archive Browser** view locks the archive file. It is then automatically unlocked when the **Archive Browser** view is closed.

Important: If a file is not recognized by Oxygen as a supported archive type, you can add it from the *Archive preferences page*.

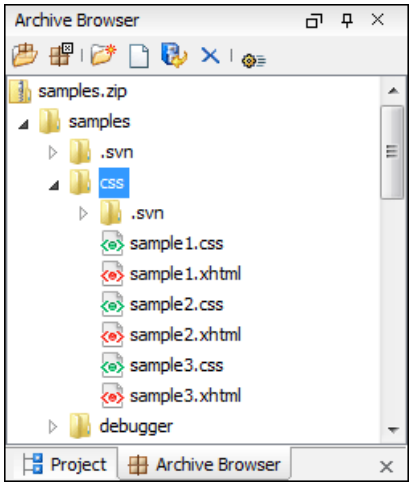


Figure 262: Browsing an Archive

The following operations are available on the **Archive Browser** toolbar:

- **Open Archive** menu - Provides access to the **Open Archive...** action which opens a new archive in the browser. If the extension is not known as an archive extension, you will be directed to the *Archive preferences page* to add a new extension. The sub-menu keeps a list of recently open archive files and a **Clear history** action which allows you to delete the list.
- **Close** - Closes the browsed archive and unlocks the archive file.
- **New folder...** - Creates a new folder as child of the selected folder in the browsed archive.
- **New file...** - Creates a new file as child of the selected folder in the browsed archive.
- **Add files...** - Adds already 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 and dropping them in the **Archive Browser** view.

- **Delete** - Deletes the selected resource in the browsed archive.
- **Archive Options...** - Opens the *Archive preferences page*.

The following additional operations are available from the **Archive Browser** contextual menu:

- **Open** - Opens a resource from the archive in the editor.
- **Extract...** - Extracts a resource from the archive in a specified folder.
- **New folder...** - Creates a new folder as child of the selected folder in the browsed archive.

- **New file...** - Creates a new file as child of the selected folder in the browsed archive.
- **Add files...** - Adds already 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.

- **Rename...** - Renames a resource in the archive.
- **Find/Replace in Files** - Allows you to search for and replace specific pieces of text inside the archive.
- **Preview** - Previews an image contained in the archive See the *Image Preview* section for more details.
- **Copy location** - Copies the URL location of the selected resource.
- **Refresh** - Refreshes the selected resource.
- **Properties** - Views properties for the selected resource.

Working with EPUB

EPUB is a free and open electronic book standard by the International Digital Publishing Forum (IDPF). It was designed for *reflowable content*, meaning that the text display can be optimized for the particular display device used by the reader of the EPUB-formatted book.

Oxygen opens EPUB files in the **Archive Browser** view, exposing all their internal bits and pieces:

- document content (XHTML and image files);
- packaging files;
- container files.

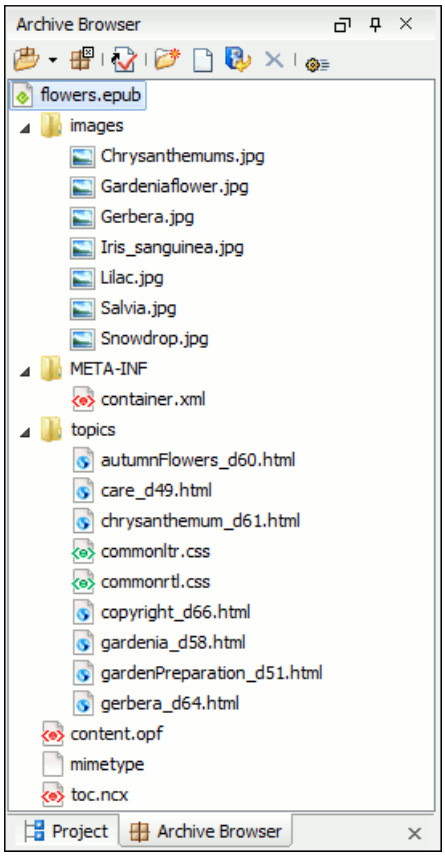


Figure 263: EPUB file displayed in the Archive Browser view

Here you can edit, delete and add files that compose the EPUB structure. To check that the EPUB file you are currently working is valid, invoke the **Validate and Check for Completeness** action. To perform the operation, Oxygen uses the open-source *EpubCheck* validator which 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.

Create an EPUB

To begin writing an EPUB file from scratch, do the following:

1. Select **File** > **New (Ctrl+N)** or press the **New** toolbar button.
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 comes with built-in support for publishing Docbook and DITA XML documents directly to EPUB.

1. Open the **Configure Transformation Scenario** dialog and choose a predefined transformation scenario.
2. Start the transformation scenario.

Editing Files From Archives

You can open in Oxygen and edit files directly from an archive using the **Archive Browser** view. When saving the archived file you will be prompted with some backup operations which can be performed to ensure that your archive data will not be corrupted. You have the following backup before save options:

- **No backup** - No backups are made.
- **Single file backup** - When you modify an archive, its content is backed up under the name `originalArchiveFileName.bak`. You can find the backup file in the same folder as the original archive.

  Note: The backup is done only once per application session for each archive open in the **Archive Browser** view.

- **Incremental backup** - When you modify an archive, its content is backed up under the name `originalArchiveFileName.bakNumber`. *Number* is an incremental integer, indicating how many backups were made so far. You can find the backup file in the same folder as the original archive.

  Note: The backup is done only once per application session for each archive open in the **Archive Browser** view.

- **Never ask me again** - Check this option if you do not want to be notified again to backup. The last backup option you chose will always be used as the default one. You can re-enable the dialog pop-up from the *Messages preferences page*.

Chapter 16

Working with Databases

Topics:

- *Relational Database Support*
- *Native XML Database (NXD) Support*
- *XQuery and Databases*
- *WebDAV Connection*

XML is a storage and interchange format for structured data and it is supported by all major database systems. Oxygen 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 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

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

The steps for configuring a data source for connecting to an IBM DB2 server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. Click the **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.

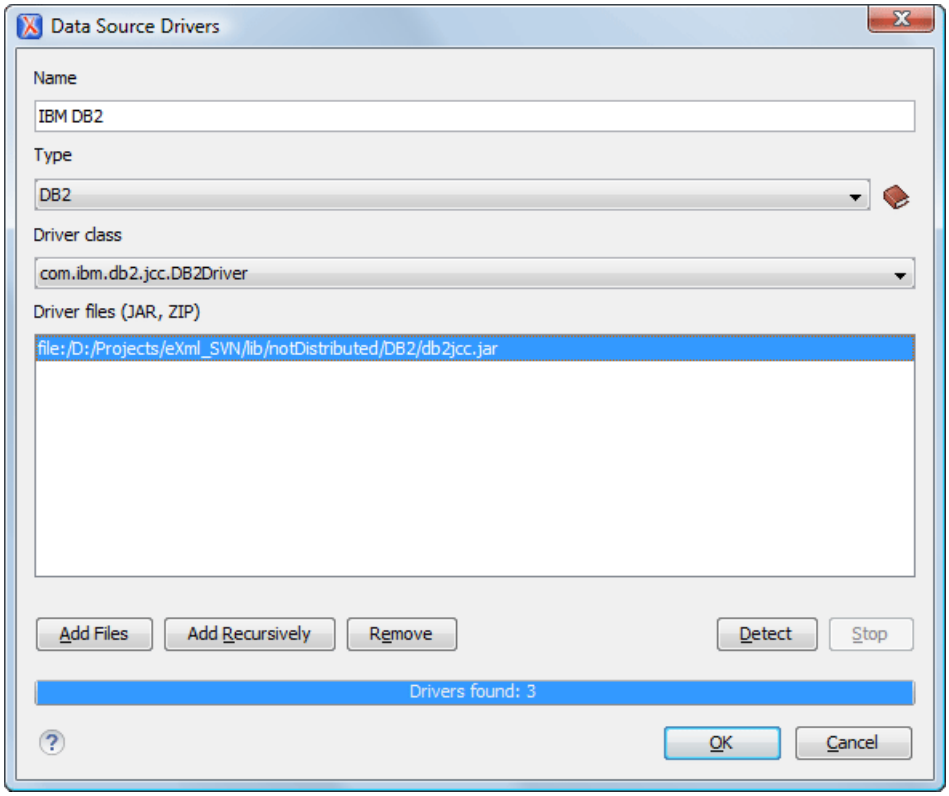


Figure 264: Data Source Drivers Configuration Dialog

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.

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

Oxygen's default configuration already contains a generic JDBC data source called **JDBC-ODBC Bridge**. Oxygen 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 menu **Preferences** > **Data Sources** .
2. Click the **New** button in the **Data Sources** panel.

   The following dialog is displayed:

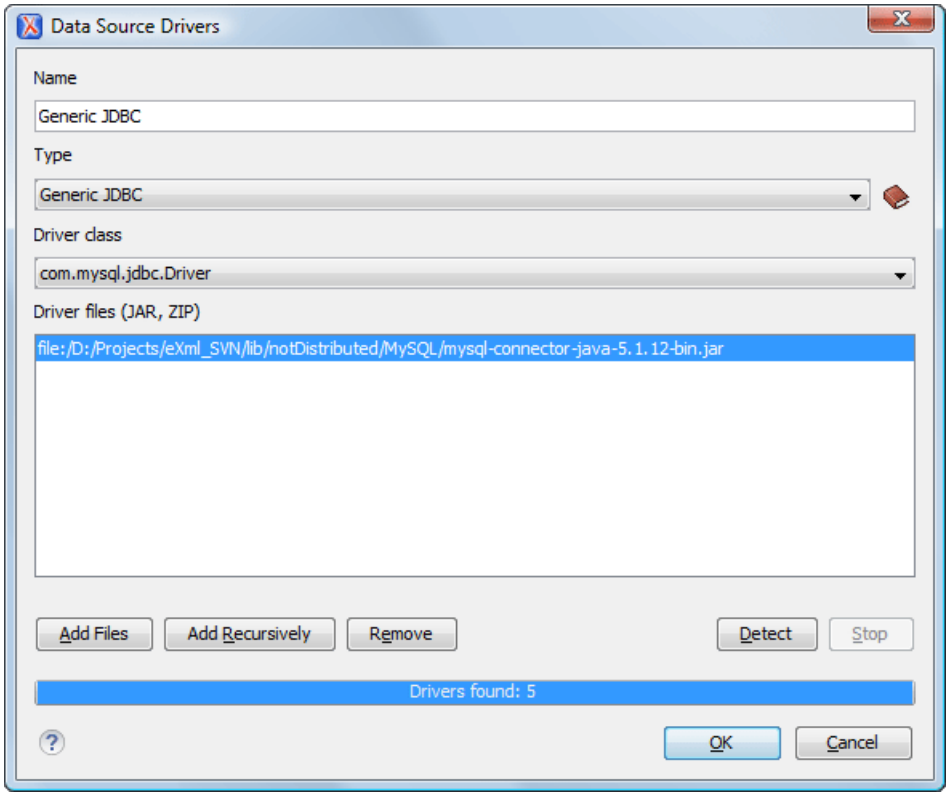


Figure 265: Data Source Drivers Configuration Dialog

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 Microsoft SQL Server Data Source

The steps for configuring a data source for connecting to a Microsoft SQL server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. Click the **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.

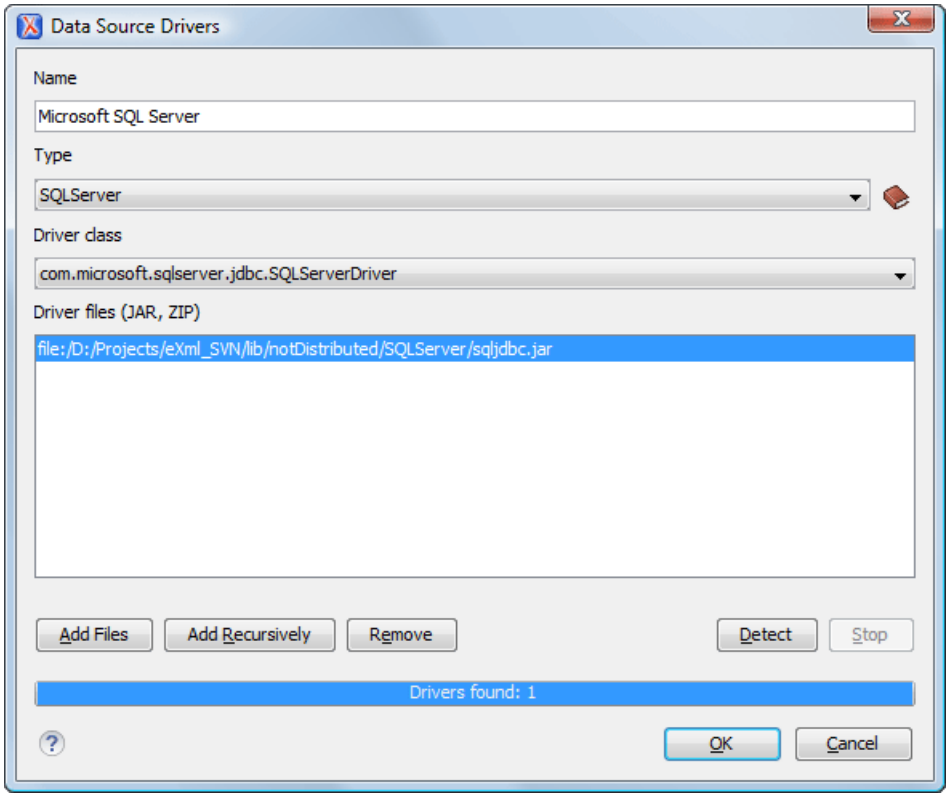


Figure 266: Data Source Drivers Configuration Dialog

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.
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

Previous versions of Oxygen (up to version 11.2) included a built-in type of data sources called **MySQL** and 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.For connecting to a MySQL server you should create a new data source of type Generic JDBC based on *the MySQL JDBC driver available from the MySQL website*. The steps for configuring such a data source are the following:

1. Go to menu **Preferences** > **Data Sources** .

2. Click the **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.

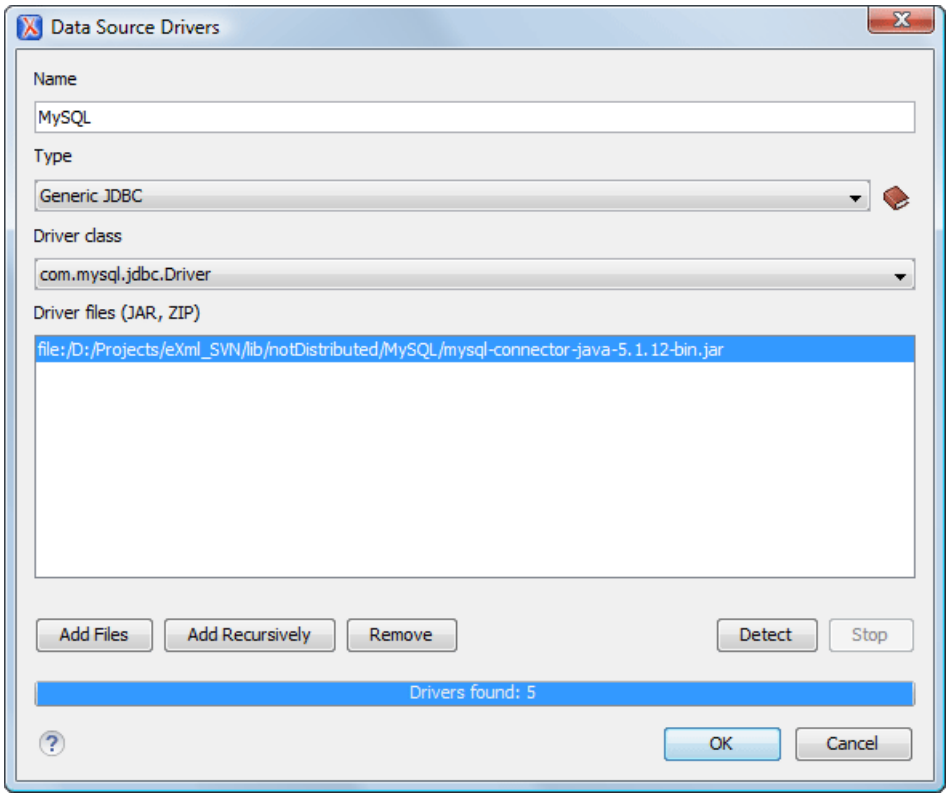


Figure 267: Data Source Drivers Configuration Dialog

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.
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

The steps for configuring a data source for connecting to an Oracle 11g server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. Click the **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.

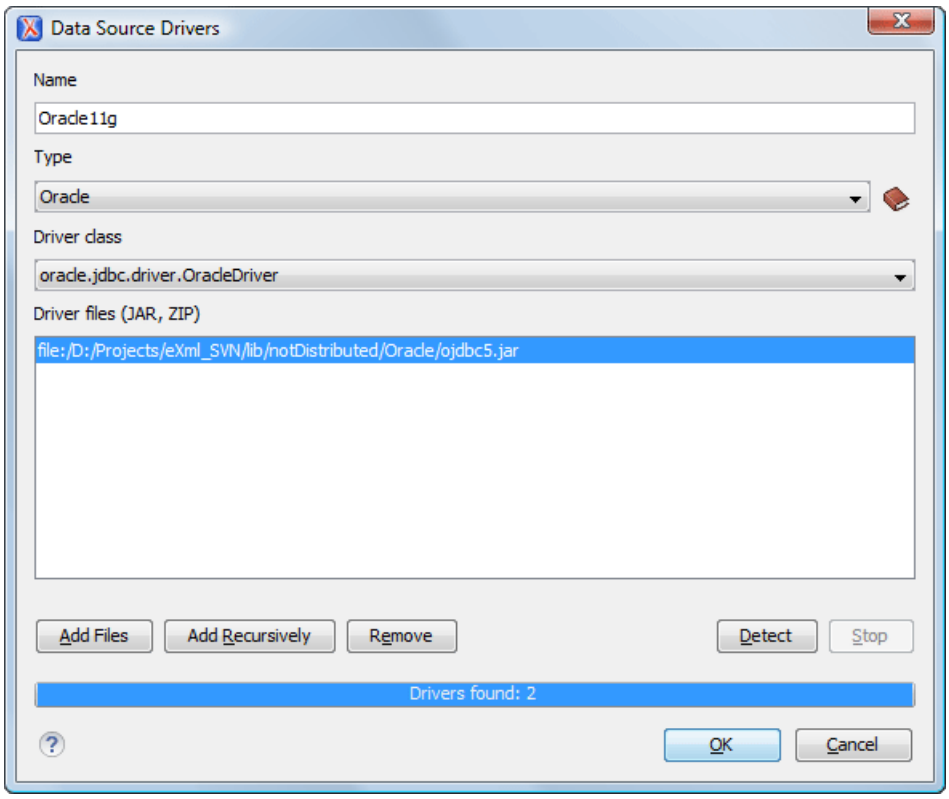


Figure 268: Data Source Drivers Configuration Dialog

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.
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 menu **Preferences** > **Data Sources** .
2. Click the **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.
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.
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

Available in the Enterprise edition only.

The steps for configuring a connection to an IBM DB2 server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. In the **Connections** panel click the **New** button.

   The dialog for configuring a database connection will be displayed.

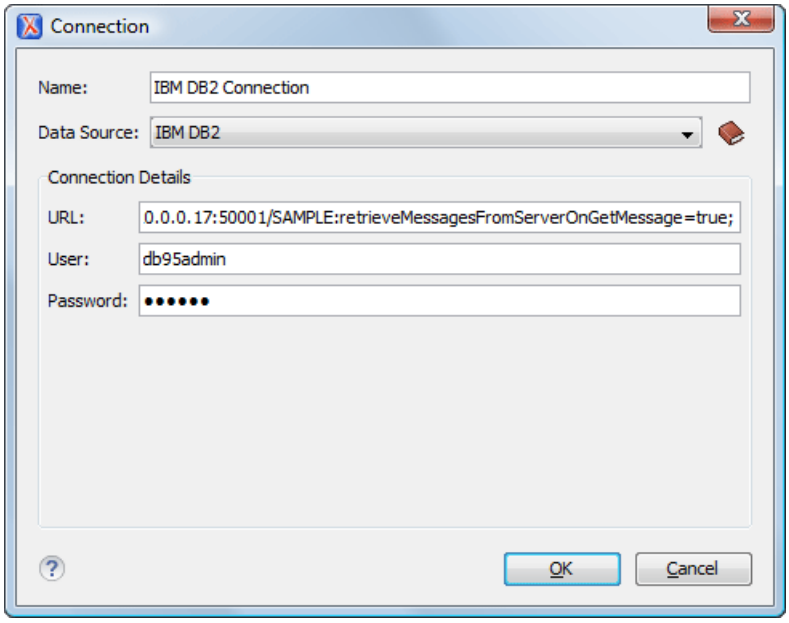


Figure 269: The Connection Configuration Dialog

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.

How to Configure a JDBC-ODBC Connection

The steps for configuring a connection to an ODBC data source are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. In the **Connections** panel click the **New** button.

   The dialog for configuring a database connection will be displayed.

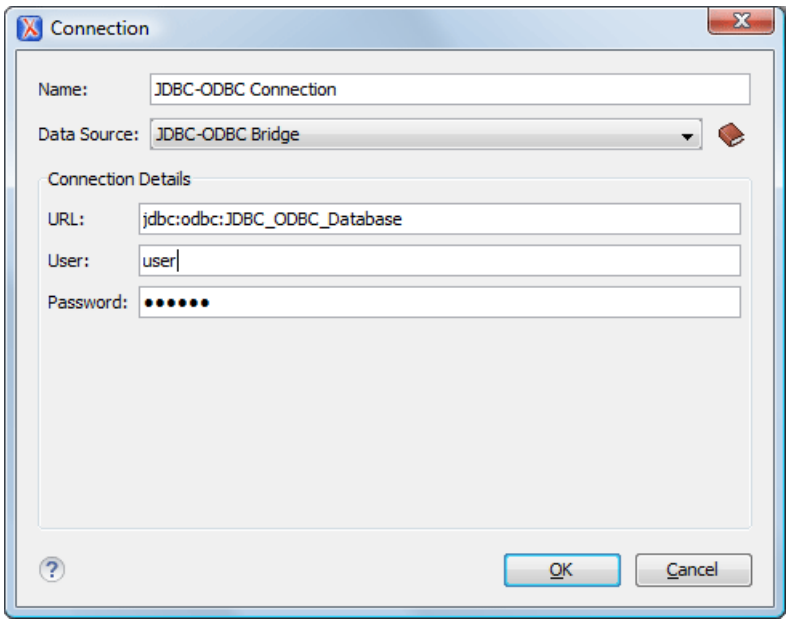


Figure 270: The Connection Configuration Dialog

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

Available in the Enterprise edition only.

The steps for configuring a connection to a Microsoft SQL Server server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. In the **Connections** panel click the **New** button.

   The dialog for configuring a database connection will be displayed.

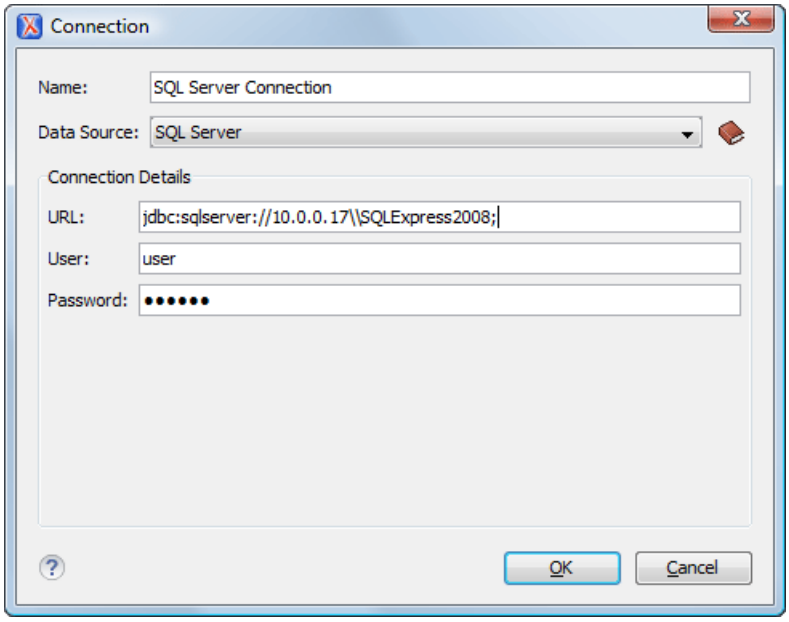


Figure 271: The Connection Configuration Dialog

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.
   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.

How to Configure a MySQL Connection

The steps for configuring a connection to a MySQL server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. In the **Connections** panel click the **New** button.

   The dialog for configuring a database connection will be displayed.

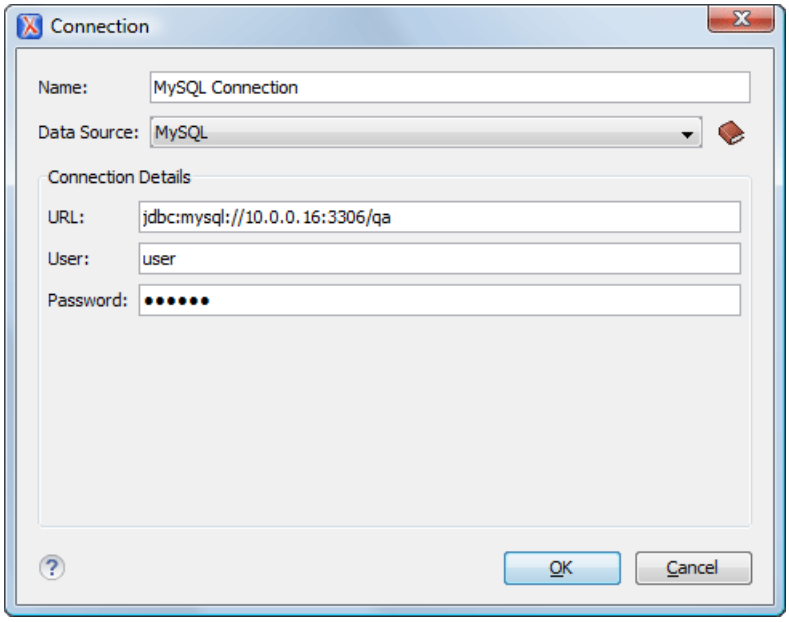


Figure 272: The Connection Configuration Dialog

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 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 menu **Preferences** > **Data Sources** .
2. In the **Connections** panel click the **New** button.

   The dialog for configuring a database connection will be displayed.

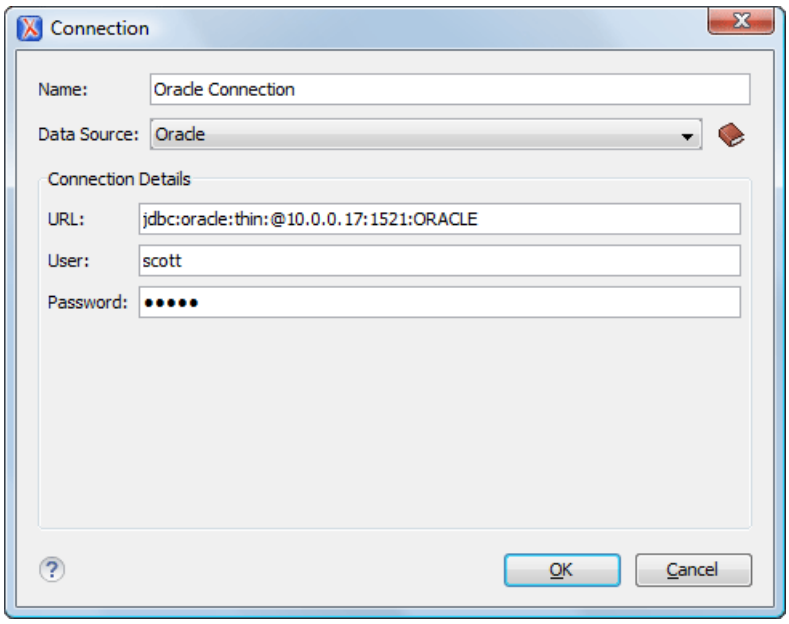


Figure 273: The Connection Configuration Dialog

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

Available in the Enterprise edition only.

The steps for configuring a connection to a PostgreSQL 8.3 server are the following:

1. Go to menu **Preferences** > **Data Sources** .
2. In the **Connections** panel click the **New** button.

   The dialog for configuring a database connection will be displayed.

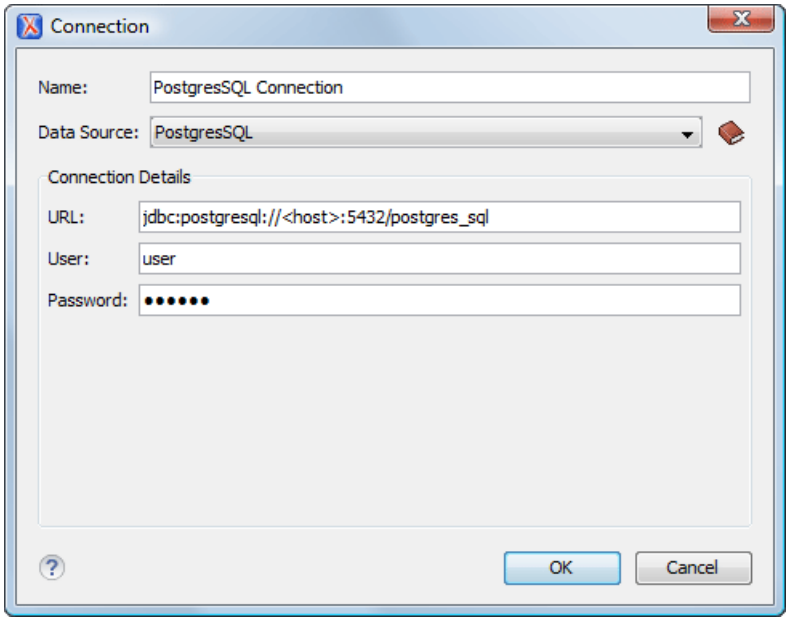


Figure 274: The Connection Configuration Dialog

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 from menu **Options** > **Preferences** > **Data Sources** . You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. Oxygen supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.

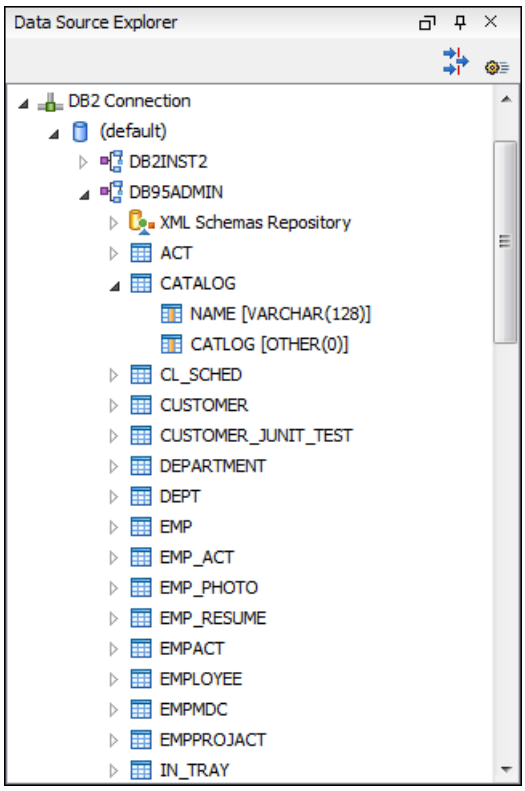


Figure 275: Data Source Explorer View

The following objects are displayed by 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**

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.

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.

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:

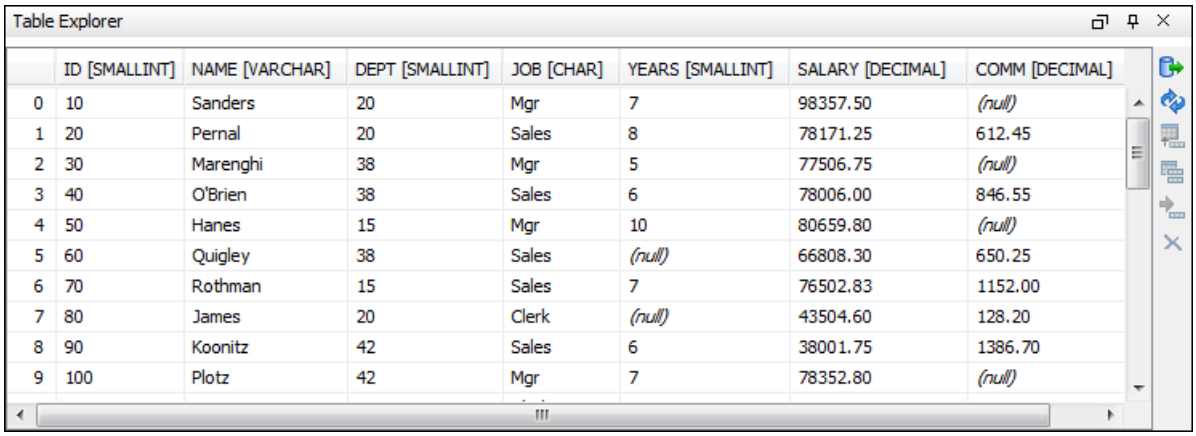
- **Refresh** - Performs a refresh of the selected node's subtree.
- **Register** - Opens a dialog for adding a new schema file in the DB XML repository. In this dialog you enter a collection name and the necessary schema files. XML Schema files management is done by using the **Add** and **Remove** 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.

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 will try to update the database with the new cell content.



	ID [SMALLINT]	NAME [VARCHAR]	DEPT [SMALLINT]	JOB [CHAR]	YEARS [SMALLINT]	SALARY [DECIMAL]	COMM [DECIMAL]
0	10	Sanders	20	Mgr	7	98357.50	*(null)*
1	20	Pernal	20	Sales	8	78171.25	612.45
2	30	Marenghi	38	Mgr	5	77506.75	*(null)*
3	40	O'Brien	38	Sales	6	78006.00	846.55
4	50	Hanes	15	Mgr	10	80659.80	*(null)*
5	60	Quigley	38	Sales	*(null)*	66808.30	650.25
6	70	Rothman	15	Sales	7	76502.83	1152.00
7	80	James	20	Clerk	*(null)*	43504.60	128.20
8	90	Koonitz	42	Sales	6	38001.75	1386.70
9	100	Plotz	42	Mgr	7	78352.80	*(null)*

Figure 276: 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'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, the cell will be marked by a red square and it will remain in editing state until a correct value is inserted. For example, in the following figure `propID` contains `LONG` values. If a character or string was inserted, the cell will look like this:

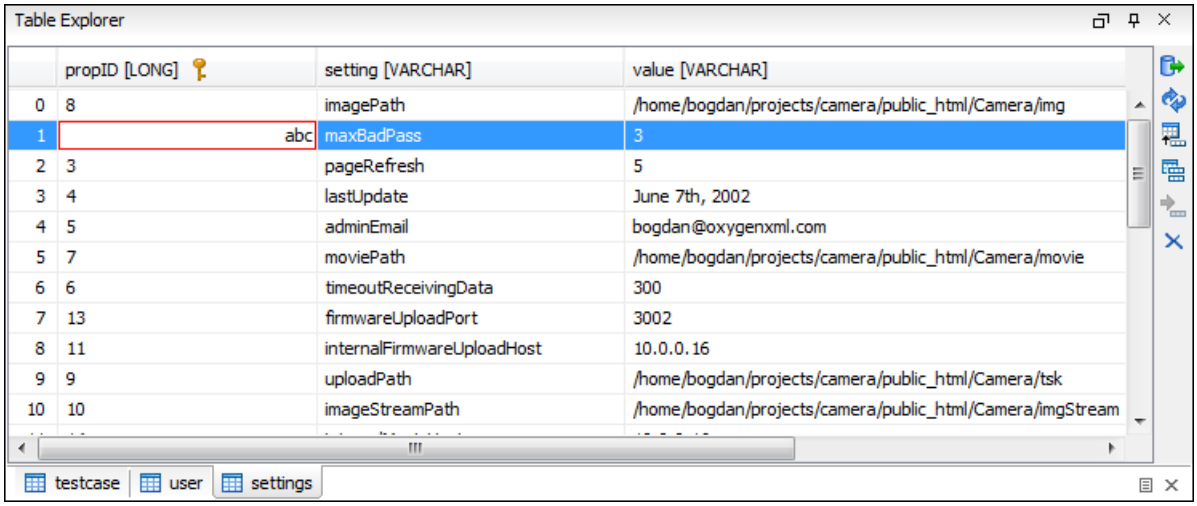


Figure 277: Cell containing an invalid value.

- 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:

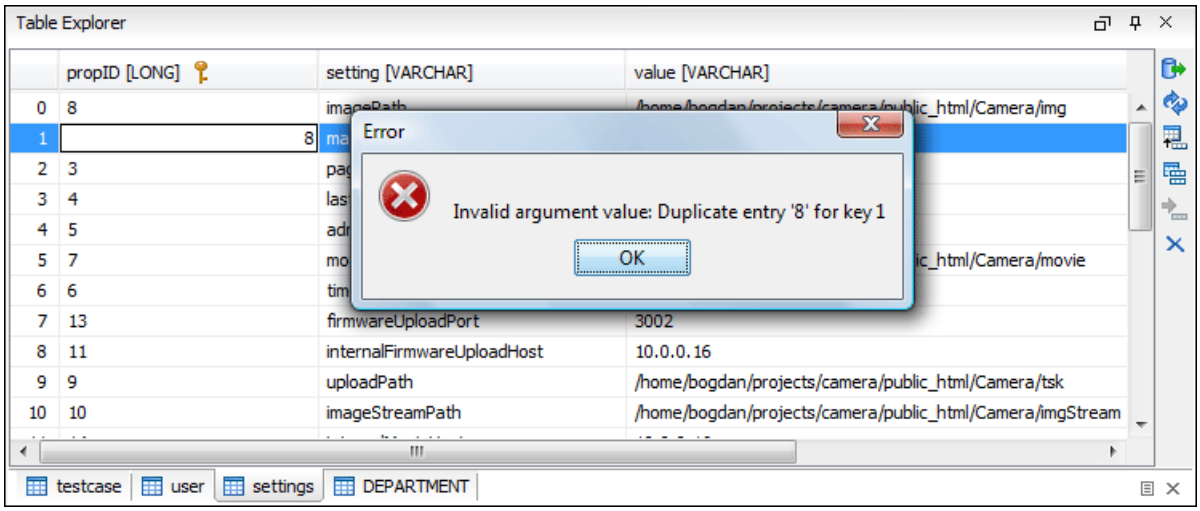


Figure 278: Duplicate entry for primary key

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'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.

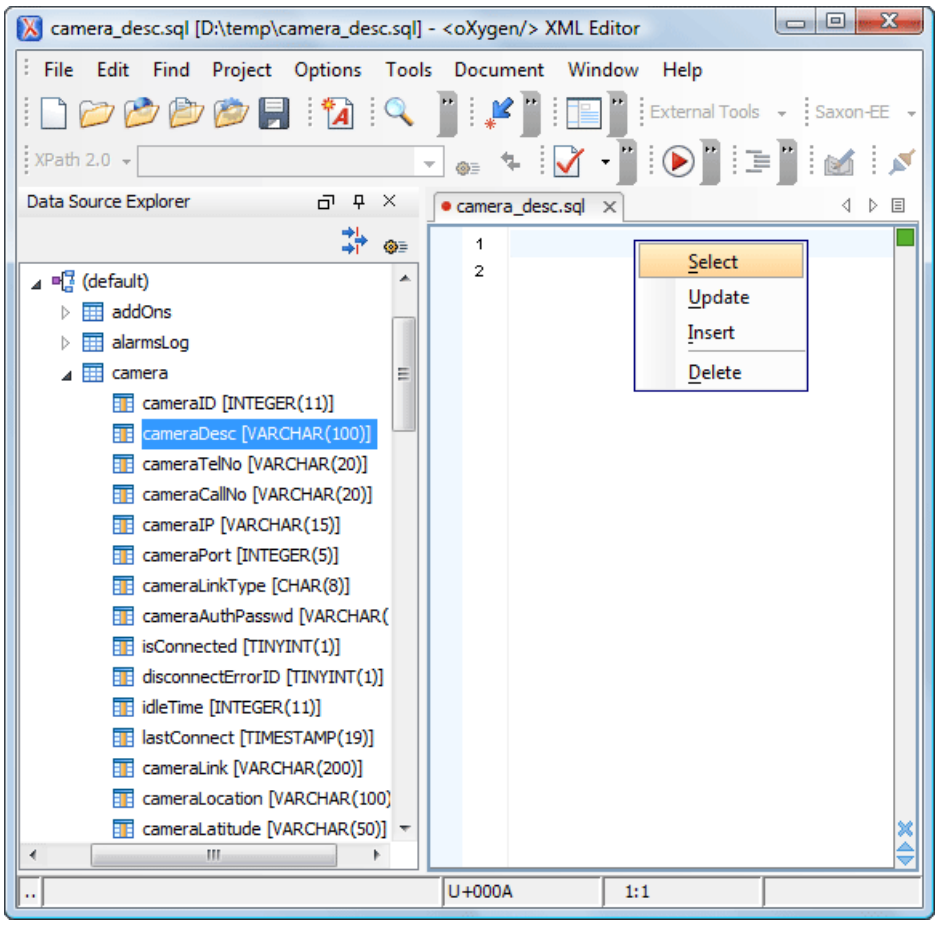


Figure 279: SQL statement editing with DND

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 `test`.`department` )`
   - UPDATE `*catalog*`.`*table*` SET `*field1*`=, `*field2*`=,.... (for this example: `UPDATE `test`.`department` SET `DEPT`=, `DEPTNAME`=, `LOCATION`=)`
   - INSERT INTO`*catalog*`.`*table*` (`*field1*`,`*field2*`, ...) VALUES (, ,) (for this example: `INSERT INTO `test`.`department` (`DEPT`,`DEPTNAME`,`LOCATION`) VALUES (, , ))`
   - DELETE FROM `*catalog*`.`*table*` (for this example: `DELETE FROM `test`.`department`)`

   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 `test`.`department` )`
   - UPDATE `*catalog*`.`*table*` SET `*field*`= (for this example: `UPDATE `test`.`department` SET `DEPT`=)`
   - INSERT INTO`*catalog*`.`*table*` (`*field1*`) VALUES () (for this example: `INSERT INTO `test`.`department` (`DEPT`) VALUES ())`
   - DELETE FROM `*catalog*`.`*table*` (for this example: `DELETE FROM `test`.`department` 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 the **Table Explorer** view at the bottom of the Oxygen 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 .

Native XML Database (NXD) Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. Oxygen offers support for the following native XML databases:

- Berkeley DB XML
- eXist
- MarkLogic
- Software AG Tamino
- Raining Data TigerLogic
- Documentum xDb (X-Hive/DB) 10
- Oracle XML DB

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 can be found on our *website*.

Oxygen supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The following directory definitions shall apply:

- OXY_DIR - Oxygen installation root directory. (for example on Windows C:\Program Files\Oxygen 12.2)
- DBXML_DIR - Berkeley DB XML database root directory. (for example on Windows C:\Program Files\Oracle\Berkeley DB XML <*version*>)
- DBXML_LIBRARY_DIR (usually on Mac and Unix is `DBXML_DIR / lib` and on Windows is `DBXML_DIR / bin`)

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`)

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 can be found on our *website*.

Oxygen 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/xmldb.jar`
   - `lib/core/xmlrpc-client-3.1.1.jar`
   - `lib/core/xmlrpc-common-3.1.1.jar`
   - `lib/core/ws-commons-util-1.0.2.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 data source configuration.

How to Configure a MarkLogic Data Source

The latest instructions on how to configure MarkLogic support in Oxygen 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 **Driver type** combo box.
5. Press the **Add** button to add the MarkLogic driver files.

   The driver file for the MarkLogic database is called `xcc.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing MarkLogic databases in Oxygen.
6. Click the **OK** button to finish the data source configuration.

How to Configure a Software AG Tamino Data Source

The latest instructions on how to configure Software AG Tamino support in Oxygen 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 *Tamino* from the **Driver type** combo box.
5. Press the **Add** button to add the Tamino driver files.

   The driver files for the Tamino database are the following:

   - `TaminoAPI4J.jar`
   - `TaminoAPI4J-l10n.jar`
   - `TaminoJCA.jar`

   👉 **Note:** You must use the jar files from the version 4.4.1 of the Tamino database.
6. Click the **OK** button to finish the data source configuration.

How to Configure a Raining Data TigerLogic Data Source

The latest instructions on how to configure TigerLogic support in Oxygen 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 *TigerLogic* from the **Driver type** combo box.
5. Press the **Add** button to add the TigerLogic driver files.

   The driver files for the TigerLogic database are found in the TigerLogic JDK lib directory from the server side:

   - `connector.jar`
   - `jca-connector.jar`
   - `tlapi.jar`
   - `tlerror.jar`
   - `utility.jar`
   - `xmlparser.jar`
   - `xmltypes.jar`

6. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source

The latest instructions on how to configure support for Documentum xDb (X-Hive/DB) 10 (X-Hive/DB version 9) in Oxygen 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 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 home directory in the **Environment home directory** field.
   b) Select the **Verbosity** level: DEBUG, INFO, WARNING or ERROR.
   c) Optionally, you can select the checkbox **Join existing environment**.

      If checked, an attempt will be made to join an existing environment in the specified home directory and all the original environment settings will be preserved. If that fails, you should consider 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.

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.

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 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 the **Port** field.
   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.
6. Click the **OK** button to finish the connection configuration.

How to Configure a Software AG Tamino Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a Tamino 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 Tamino engine in the **XML DB URI** field.
   b) Set the user name to access the Tamino engine in the **User** field.
   c) Set the password to access the Tamino engine in the **Password** field.
   d) Set the name of the database to access from the Tamino engine in the **Database** field.
   e) Check the checkbox **Show system collections** if you want to see the Tamino system collections in the **Data Source Explorer** view.
6. Click the **OK** button to finish the connection configuration.

How to Configure a Raining Data TigerLogic Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a TigerLogic 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 host name or IP address of the TigerLogic engine in the **Host** field.
   b) Set the port number of the TigerLogic engine in the **Port** field.
   c) Set the user name to access the TigerLogic engine in the **User** field.
   d) Set the password to access the TigerLogic engine in the **Password** field.
   e) Set the name of the database to access from the TigerLogic database engine in the **Database** field.
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. 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 from menu **Options** > **Preferences** > **Data Sources** . You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. Oxygen supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.

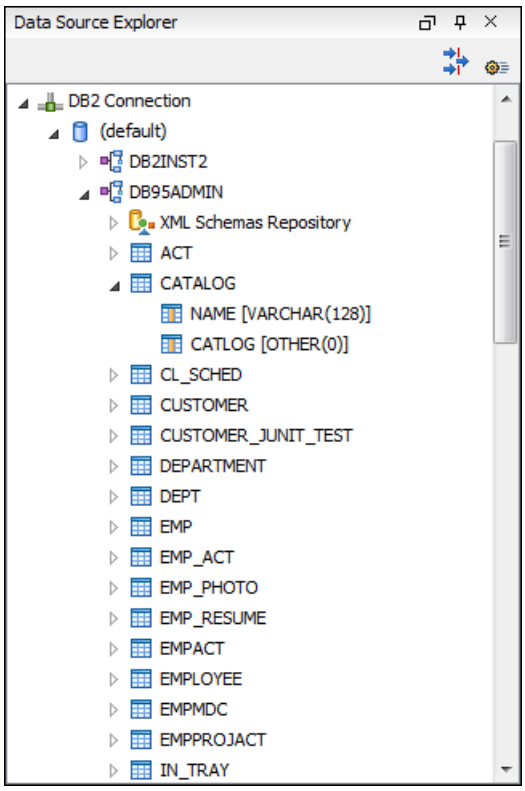


Figure 280: Data Source Explorer View

The following objects are displayed by 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**

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 allows the user to browse the native Oracle XML Repository and perform various operations on the resources in the repository.

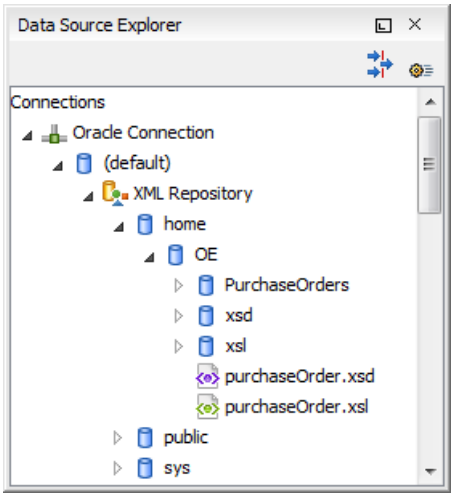


Figure 281: Browsing the Oracle XML DB Repository

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.

For running XQuery transformation on collections from XML Repository please see *a tutorial from Oracle*.

PostgreSQL Connection

Oxygen 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.

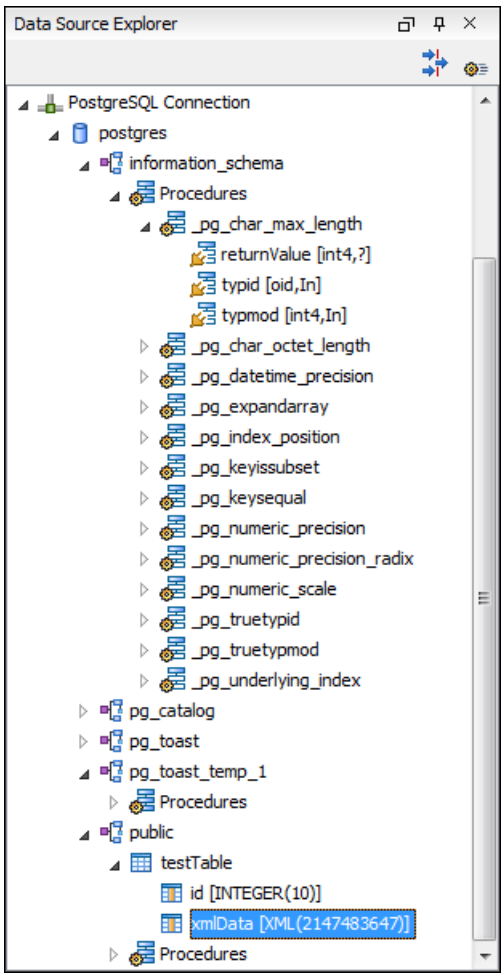


Figure 282: Browsing a PostgreSQL repository

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*.

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.

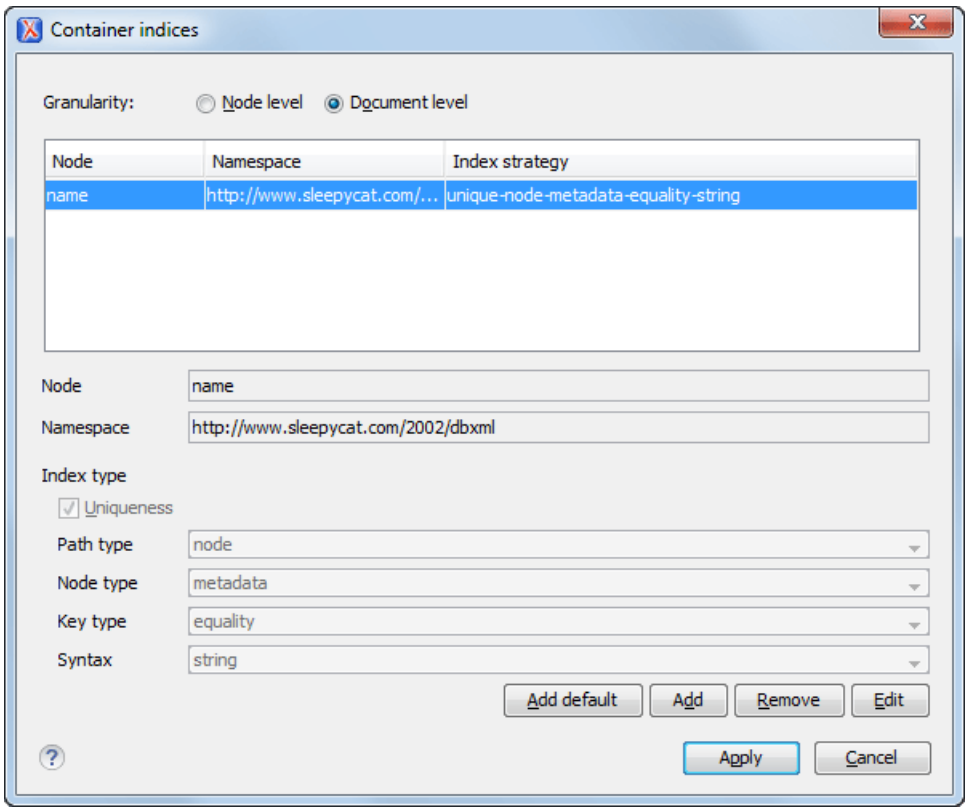


Figure 283: Container indices

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.

- **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.

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.

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:

- **Refresh** - Performs a refresh of the selected node's subtree.
- **New Collection** - Creates a new 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.
- **Delete** - Removes the selected collection.
- **Rename** - Allows you to change the name of the selected collection.
- **Move** - Allows you to move the selected collection in a different location in the database tree (also available through drag and drop).

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.

- **Move** - Allows you to move the selected resource in a different collection in the database tree (also available through drag and drop).
- **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.

MarkLogic Connection

The resource management for a MarkLogic database ca be done through WebDAV. For this a WebDAV URL must be configured in *the MarkLogic connection*. The actions that can be performed on MarkLogic resources through WebDAV are the same used for a WebDAV connection (see more about this in *WebDAV Connection* section).

Software AG Tamino Connection

This section explains the actions that are available on a Tamino connection.

Actions Available at Connection Level

For a Tamino 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 container** - Allows you to create a new collection in the database.

Actions Available at Collection Level

For every new Tamino collection created in the **Data Source Explorer** view, you can specify if a schema is *required*, *optional* or *prohibited*. The following actions are available at collection level:

- **Refresh** - Performs a refresh of the selected node's subtree.
- **Filter ...** - An XQuery expression can be specified for filtering the nodes displayed in the selected Tamino container. It is only possible to specify one predicate. In the XQuery syntax a predicate is enclosed in square brackets. The square brackets, however, must not be specified in the dialog box displayed by this action. Only the predicate must be specified and it will be applied on the selected document type. For example: `name/surname between 'B', 'C'`
- **Insert XML instance** - Allows you to load a new XML document.
- **Insert non XML instance** - Allows you to load a non XML document.
- **Modify Collection Properties** - Allows you to change the schema usage for the selected collection to optional. This action is available on collections with required and prohibited schema usage.
- **Define schema** - Allows you to add a new schema in the Schema Repository. This action is available on collections with optional and required schema usage.
- **Delete** - Removes the selected collection. If it is a Tamino doctype then the action removes all the XML instances contained in the document type.
- **Set default** - Sets this collection as the default collection for running queries with the `input()` function.

Actions Available at Schema Level

For a Tamino database the actions available at schema level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected schema.
- **Open** - Opens the selected schema in the editor. There are supported schema changes that preserve the validity relative to the existent instances.
- **Delete** - Removes the selected schema from the Schema Repository.

Actions Available at Resource Level

For a Tamino 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.
- **Delete** - Removes the selected 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** - 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.

Validation of an XML resource stored in a Tamino database is done against the schema associated with the resource in the database.

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).
- **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.

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.

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 offer XQuery support:

- *Native XML Databases:*
  - Berkeley DB XML
  - eXist
  - MarkLogic (validation support not available)
  - Software AG Tamino
  - Raining Data TigerLogic (validation support not available)
  - 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.

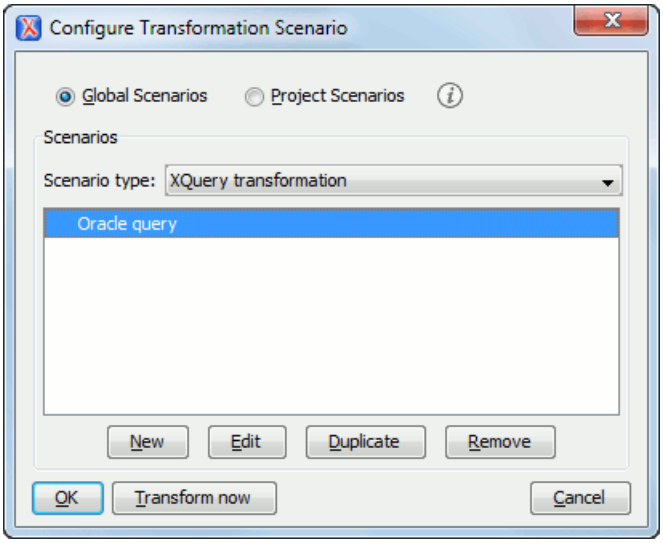
1. Configure a data source for the database.

   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.

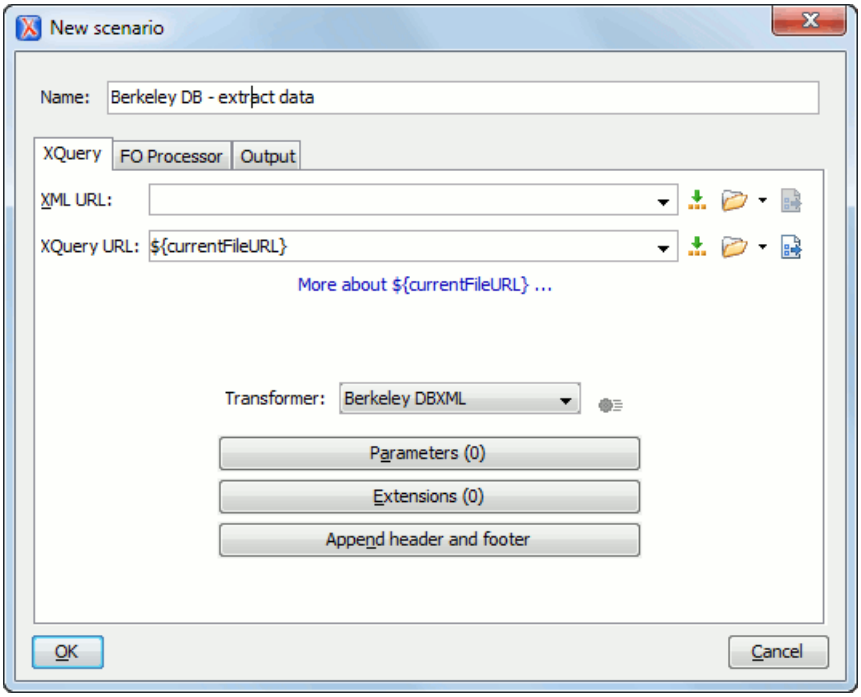


Figure 284: Edit Scenario Dialog

   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 window, for example an XMLTYPE value or a CLOB value, do the following actions:

- right click on that table cell
- select the action **Copy cell** 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 .

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 you will first need to configure a *MarkLogic data source* and a *MarkLogic connection*. Also you have to make sure that the debugging support is enabled in the MarkLogic server that will be accessed from Oxygen. On the server side debugging must be activated both in the XDBC server and in the section *Task Server* of the server control console (the switch *debug allow*) otherwise the error `DBG-TASKDEBUGALLOW` is reported by the MarkLogic server.

The MarkLogic XQuery debugger integrates seamlessly into the *XQuery Debugger perspective*. If you already have a MarkLogic scenario configured for the XQuery file you can choose directly to *debug the scenario*. If not, you just have to 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.

Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is available only for MarkLogic server versions 3.2 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'
- All the debugging steps are executed by the MarkLogic server and the results or possible errors of each step are presented by the local debugger user interface.
- 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*.
- *Breakpoints* can be set in the imported modules but they are only active if the modules are opened in the editor at the time of debugging.
- The modules can only be opened in the editor during the debugging session by stepping in repeatedly until reaching the module.
- There should not be any breakpoints set 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'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

Oxygen's default configuration already contains a WebDAV data source called **WebDAV**. 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.

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* where you can configure both data sources and connections.
- **Add Resource ...** - Allows you to add a new file on the server.
- **Add Container ...** - Allows you to create a new folder on the server.
- **Refresh** - Performs a refresh of the connection.

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:

- **Add Container** - Allows you to create a new folder on the server.
- **Add Resource** - Allows you to add a new file on the server in the current folder.
- **Rename** - Allows you to change the name of the selected folder.
- **Move** - Allows you to move the selected folder in a different location in the tree (also available through drag and drop).
- **Delete** - Removes the selected folder.
- **Refresh** - Performs a refresh of the selected node's subtree.

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.
- **Unlock** - Removes the lock from the current file in the database.
- **Rename** - Allows you to change the name of the selected file.
- **Move** - Allows you to move the selected file in a different location in the tree (also available through drag and drop).
- **Delete** - Removes the selected file.
- **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 node.
- **Properties** - Displays the properties of the current file in a dialog.

Chapter 17

Importing Data

Topics:

- *Introduction*
- *Import from Database*
- *Import from MS Excel Files*
- *Import from HTML Files*
- *Import from Text Files*

This chapter shows you how to import data that is stored in text format, in an Excel sheet or in relational database tables into XML documents.

Introduction

XML was designed to describe data. 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 many different types of applications.

This is why Oxygen now offers you support for importing text files, MS Excel files, Database Data and HTML files into XML documents. The XML documents can be further converted into other formats using the Transform features.

Import from Database

This section explains how to import data from a database into Oxygen.

Import Table Content as XML Document

The steps for importing the data of a table from a relational database are the following:

1. Go to menu **File** > **Import** > **Database Data...**

   This steps opens a dialog with all the database connections:

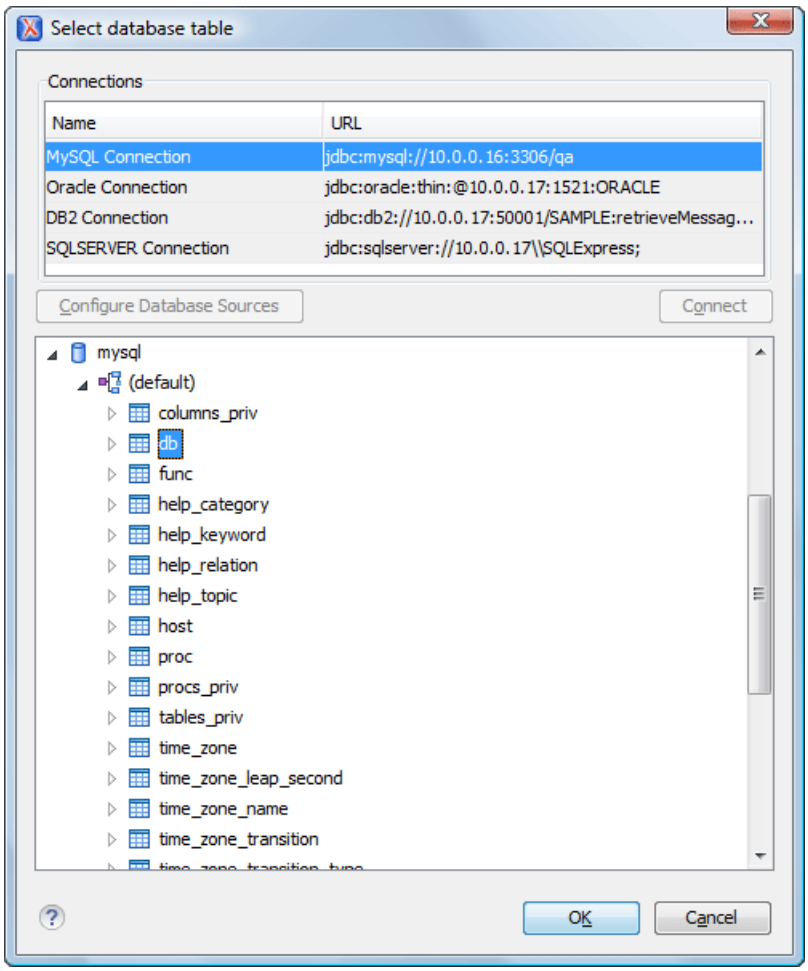


Figure 285: Select Database Table Dialog

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** dialog will be opened at **Data Sources** section.
4. Click the **Connect** button.
5. From the catalogs list click on a schema and choose the required table.
6. Click the **OK** button.

   The **Import Criteria** dialog will open next, showing a default query string like `select * from table` in the **SQL Query** pane:

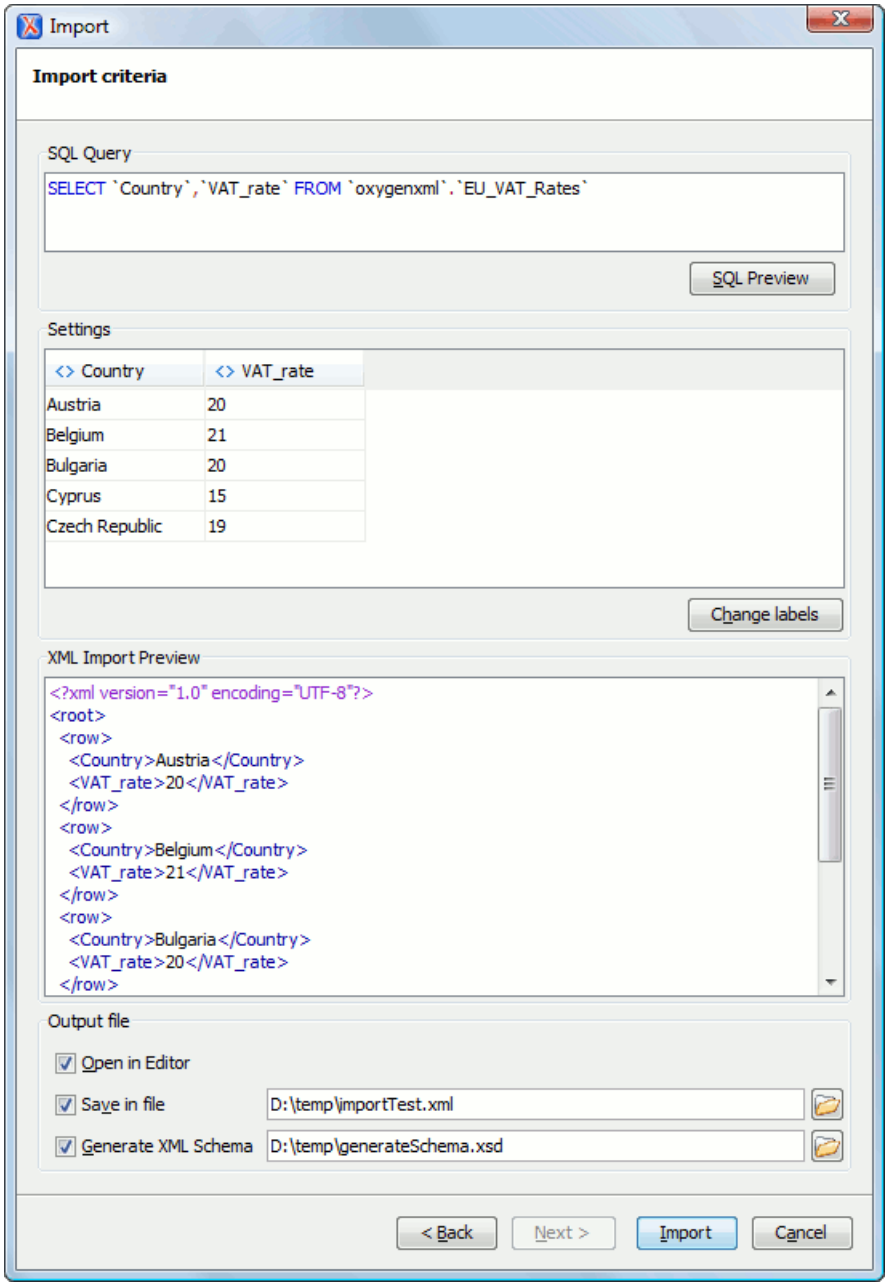


Figure 286: Import from Database Criteria Dialog

The dialog contains the following items:

- **SQL Preview** - Displays the labels that will be used in the XML document in the preview from the panel **Import settings**. If the **SQL Preview** button is pressed, it shows the labels that will be used in the XML document and the first 5 lines from the database. All data items in the input will be converted by default to element content, but this can be over-ridden by clicking on the individual column headers. Clicking once on a column header (ex **Heading0**) will cause the data from this column to be used as attribute values on the row elements. Clicking a second time - the column's data will be ignored when generating the XML file. You can cycle through these three options by continuing to click on the column header. If the data column will be converted to element content, the header will contain the <> symbol. If the data column will be converted to attribute content, the header will contain the = symbol, and if it will be skipped, the header will contain an *x*.
- **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 on 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**.
- **Open in editor** - If checked, the new XML document created from the imported text file will be opened in the editor.
- **Save in file** - If checked, the new XML document will be saved at the specified path.

  👉 **Note:** If only **Open in editor** is checked, the newly created document will be opened 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 should be taken into consideration.

   If you edit the query string so that the query does a join of two or more tables and selects columns with the same name from different tables you should use an alias for the columns like in the following example. That will avoid a confusion of two columns 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 will be displayed in a tabular form in the **Import Settings** panel. The **XML Import Preview** panel will contain an example of what the generated XML will look like.

Convert Table Structure to XML Schema

The structure of a table from a relational database can be imported in Oxygen as an XML Schema. For this the checkbox **Generate XML Schema** must be selected in the dialog **Import criteria** from *the procedure for importing table data* as an XML instance document.

Import from MS Excel Files

Oxygen can also import MS (Microsoft) Excel files into XML format documents with the following steps:

1. Go to menu **File** > **Import** > **MS Excel File...** .
   A dialog **Select Excel Sheet** will be opened.

2. Enter the URL of the Excel document in the dialog.
3. Choose one of the available sheets of the Excel document.
   The input data is displayed next in the **Import Criteria** dialog in a tabular form and the **XML Import Preview** contains an example of what the generated XML will look like. The **Import Criteria** dialog has a similar behaviour with the one shown in case of *Import from text files*.
4. Click the **OK** button.

Import from HTML Files

HTML is one of the formats that can be imported as an XML document. The steps are the following:

1. Go to menu **File** > **Import** > **HTML File ...** .
   The **Import HTML** dialog is displayed.
2. Enter the URL of the HTML document.
3. Type a name for the new document that will contain the imported XML content.
4. Select the type of the result XHTML document:
   - XHTML 1.0 Transitional
   - XHTML 1.0 Strict
5. Click the **OK** button.

The resulted document will be an XHTML file containing a DOCTYPE declaration referring to the XHTML DTD definition on the Web. The parsed content of the imported file will be transformed to XHTML Transitional or XHTML Strict depending on what radio button the user chose when performing the import operation.

Import from Text Files

The steps for importing a text file into an XML file are the following:

1. Go to menu **File** > **Import** > **Text File...**
   The **Select text file** dialog will be 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 will be displayed:

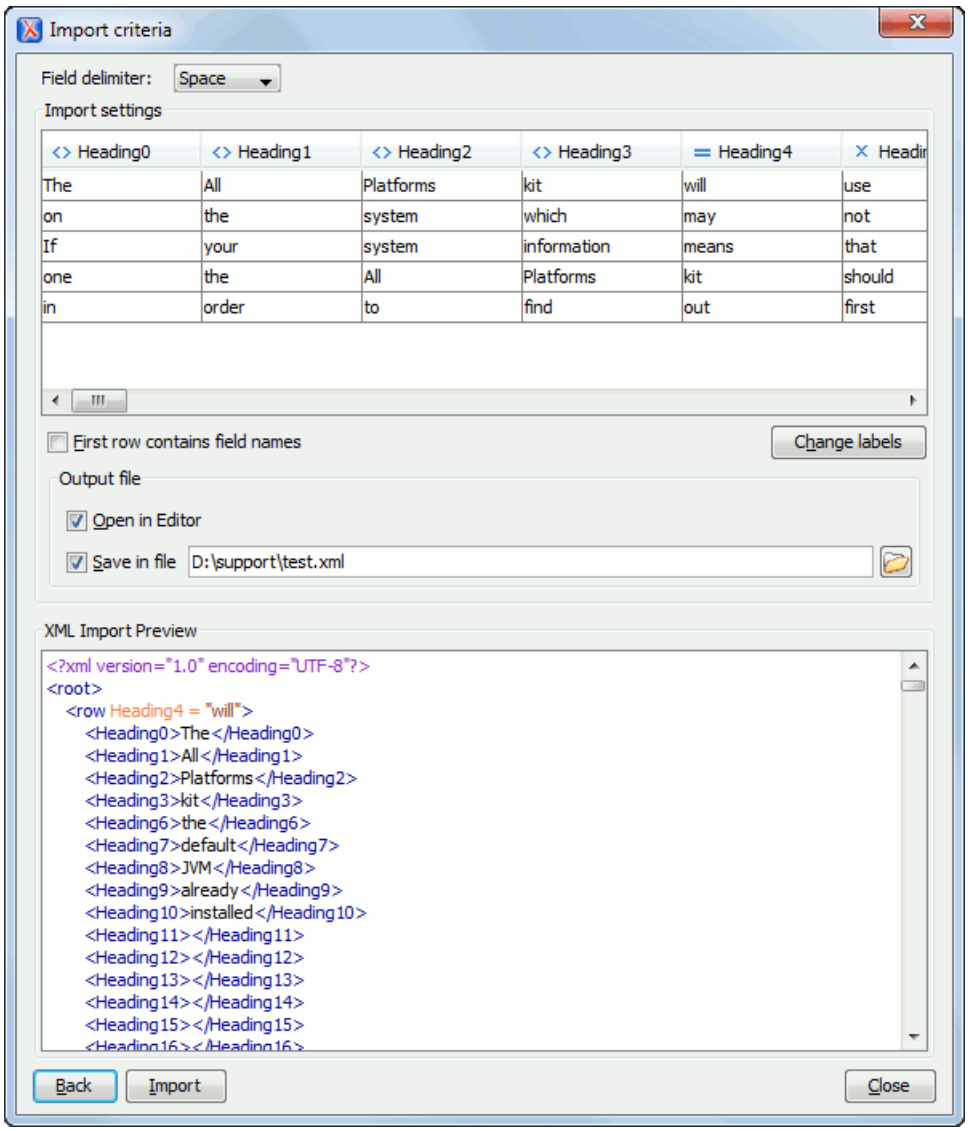


Figure 287: Import from text file

The input data is displayed in a tabular form. The **XML Import Preview** panel contains an example of what the generated XML will look like. The names of the XML elements and the transformation of the first 5 lines from the text file are displayed. All data items in the input will be converted by default to element content, but this can be over-ridden by clicking on the individual column headers. Clicking once on a column header will cause the data from this column to be used as attribute values on the row elements. Clicking a second time - the column's data will be ignored when generating the XML file. You can cycle through these three options by continuing to click on the column header. If the data column will be converted to element content, the header will contain the <> symbol. If the data column will be converted to attribute content, the header will contain the = symbol. If it will be skipped, the header will contain an *x*.

5. Select the field delimiter for the import settings:
   - comma
   - semicolon
   - tab
   - space
6. Set other optional settings of the conversion.

   The dialog offers the following settings:

- **First row contains field names** - If the option is checked, you'll notice that the table has moved up. The default column headers are replaced (where there is information) 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 (where the first row is interpreted as not containing field names), simply uncheck the option.
- **Change labels** - If the above option is set, the first row of the input file contains presentation names and these will be used as tokens in the created XML files, otherwise some generic heading names will be used. 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 on the desired item and entering the required label. The conversion criterion can also be modified by selecting from the drop-down list one of the options: **ELEMENT**, **ATTRIBUTE** or **SKIPPED**.
- **Open in editor** - If checked, the new XML document created from the imported text file will be opened in the editor.
- **Save in file** - If checked, the new XML document will be saved at the specified path.

  👉 **Note:** If only **Open in editor** is checked, the newly created document will be opened in the editor, but as an unsaved file.

Chapter 18

Content Management System (CMS) Integration

Topics:

- *Integration with Documentum (CMS)*

This chapter explains how Oxygen can be integrated with a content management system (CMS) so that the data stored in the CMS can be edited directly in the Oxygen editor. Only the integration with the Documentum (CMS) is explained but other CMSs can use the *plugin support* for similar integrations.

Integration with Documentum (CMS)

Oxygen 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 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 1.5.x or 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

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

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 at the time of this implementation there is a problem in the UCF Client implementation for MAC OS X which prevents you from viewing or editing XML documents from the repository. 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.
2. In order for the Documentum driver to work faster, 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 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 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.

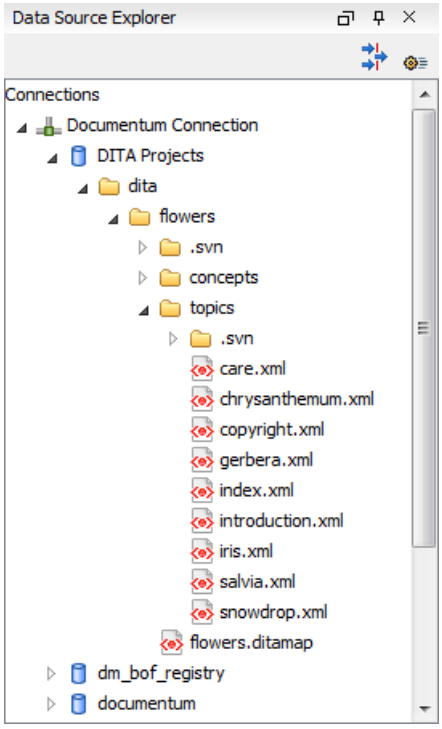


Figure 288: Browsing a Documentum repository

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)** 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 object in the editor.
- **Edit with** - Checks out (if not already checked out) and opens the selected object in the specified editor / tool.
- **Open (Read-only)** - Opens the selected object in the editor for viewing. The files are marked as read-only in the editor using a lock icon on the file tab. If you want to edit those files you must enable the *Can edit read only files* option.
- **Open with** - Opens the selected object in the specified editor / tool for viewing.
- **Check Out** - Checks out the selected object from the repository. The action is not available if the object is already checked out.
- **Check In** - Checks in the selected object (commits changes) into the repository. The action is only available if the object is checked out.

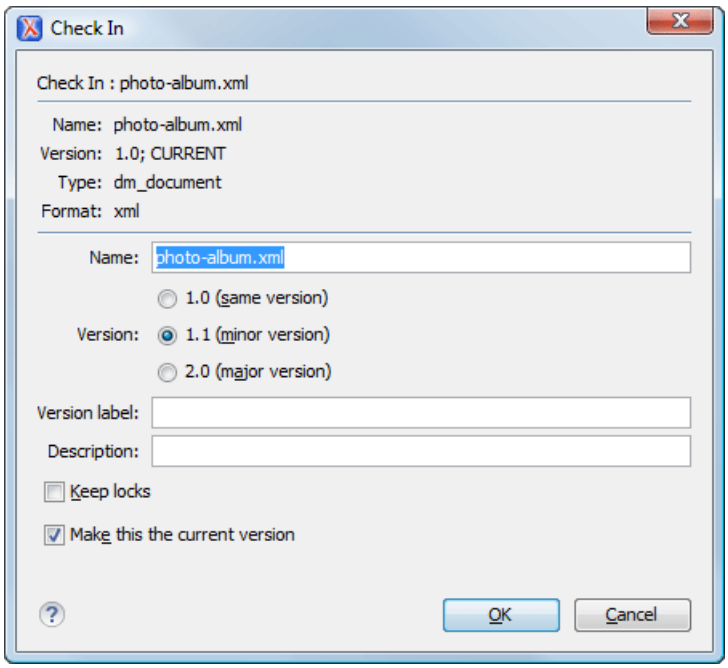


Figure 289: Check In Dialog

The properties of a resource are the following:

- **Name** - The name the file will have on the repository.
- **Version** - Allows you to choose what version the object will have after being checked in.
- **Version label** - The label of the updated version.
- **Description** - An optional description of the file.
- **Keep Locks** - If checked the updated file is checked into the repository but it is also kept checked out in your name.
- **Make this the current version** - Makes the updated file the current version (will have the *CURRENT* version label).

- **Cancel Checkout** - Cancels the check out and loses all modifications since the check out. Action is only available if the object is checked out.
- **Export** - Allows you to export the object and save it locally.
- **Rename** - Changes the name of the selected object.
- **Copy** - Copies the selected object to 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)** key pressed.
- **Move** - Moves the selected object to a different location in the tree. Action is not available on virtual document descendants and on checked out objects. This action can also be performed with drag and drop.
- **Delete** - Deletes the selected object from the repository. Action is not available on virtual document descendants and on checked out objects.
- **Add Relationship** - Adds a new relationship for the selected object. This action can also be performed with drag and drop between objects.
- **Convert to Virtual Document** - Allows you to convert a simple document to a virtual document. Action is available only if the object 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 object is a virtual document with no descendants.
- **Copy location** - Allows you to copy to clipboard an application specific URL for the object which can then be used for various actions like opening or transforming the resources.
- **Refresh** - Performs a refresh of the selected object.
- **Properties** - Displays the list of properties of the selected object.

Transformations on DITA Content from Documentum (CMS)

Oxygen 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 or a customization of such a predefined DITA transformation.

Chapter 19

Composing Web Service Calls

Topics:

- *Overview*
- *Composing a SOAP Request*
- *Testing Remote WSDL Files*
- *The UDDI Registry Browser*
- *Generate WSDL Documentation*

This chapter presents how to compose a SOAP request based on a WSDL file, how to send the request to a server and how to generate HTML documentation for WSDL files.

Overview

Web Services Description Language (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 WSDL files contain information about the published services, like the name, the message types and the bindings. The editor is offering a way to edit the WSDL files that is similar to editing XML, the content completion and validation being driven by a mix of the WSDL and SOAP schemas. Oxygen supports WSDL version 1.1 and 2.0 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 after that against a SOAP 1.2 schema. In addition to validation against the XSD schemas the WSDL file is also analysed during validation so that more element reference specific problems can be detected.

After you edit and validate your Web service descriptor against a mix of the XML Schemas for WSDL and SOAP it is very easy to check if the defined SOAP messages are accepted by the remote Web Services server using Oxygen's **WSDL SOAP Analyser** integrated tool.

Composing a SOAP Request

To design, compose, and test Web service calls in Oxygen follow the procedure:

1. *Create a new document* or *open an existing document* of type WSDL.
2. Design the Web Service descriptor in the WSDL editor.

   The *content completion* is driven by a mix of the WSDL and SOAP schemas. You do not need to specify the schema location for the WSDL standard namespaces because Oxygen comes with these schemas and uses them by default to assist the user in editing Web Service descriptors.

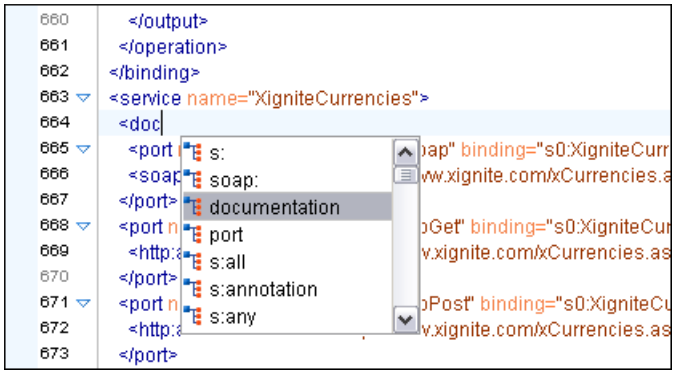


Figure 290: Content completion for WSDL documents

3. While editing the Web-Services descriptors *check their conformance* to the WSDL and SOAP schemas.

   In the following example you can see how the errors are reported.

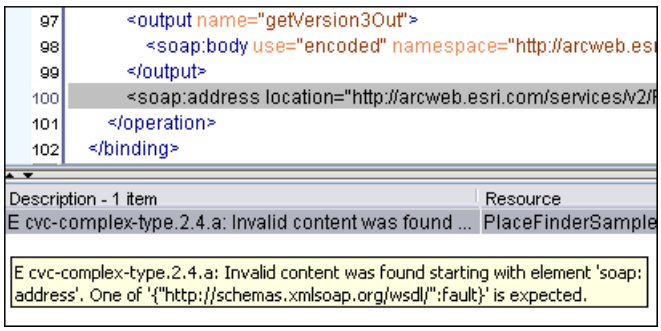


Figure 291: Validating a WSDL file

4. Check if the defined messages are accepted by the Web Services server.

   Oxygen is providing two ways of testing, one for the currently edited WSDL file and other for the remote WSDL files that are published on a web server. For the currently edited WSDL file the WSDL SOAP Analyser tool can be opened by:

   - pressing the toolbar button **WSDL SOAP Analyser**
   - going to the menu item **Document** > **Tools** > **WSDL SOAP Analyser**
   - going to submenu **Open with** > **WSDL SOAP Analyser** of the **Project** view contextual menu

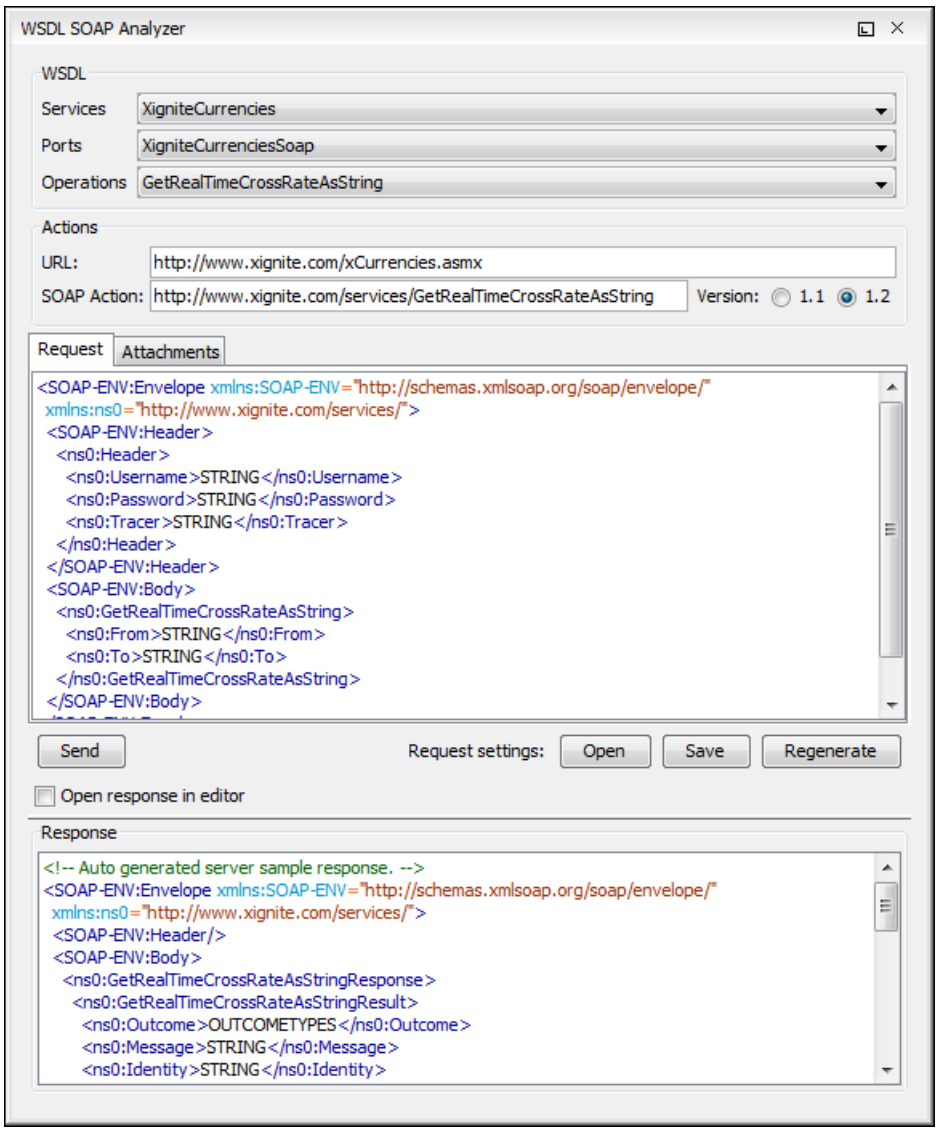


Figure 292: WSDL SOAP Analyser

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 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 is available for this editor and is driven by the schema that defines the type of the current message. While selecting different operations Oxygen will remember 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's 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 will look like. After pressing the **Send** button it will present 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 will prompt you to save them, then will try 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 will be listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog is shown when Oxygen is connecting to the server.

The testing of a WSDL file is straight-forward: just click on the WSDL analysis button, then select the service, the port and the operation. The editor will generate 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. For testing remote WSDL files *see this section*.

5. Save the request derived from the Web Service descriptor.

   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 will save time in configuring the URLs and parameters.

6. Open the result of a Web Service call in an editor panel.

   In this way you can save the SOAP request or process it further.

Testing Remote WSDL Files

To open and test a remote WSDL file the steps are the following:

1. Go to menu **Tools** > **WSDL SOAP Analyser ...** .
2. On the **WSDL File** tab enter the URL of the remote WSDL file.

   You enter the URL:

   - by typing
   - by browsing the local file system
   - by browsing a remote file system
   - by browsing *a UDDI Registry*

3. Press the **OK** button.
   This will open the **WSDL SOAP Analyser** tool. In the **Saved SOAP Request** tab you can open directly a previously saved Web Service SOAP Call (WSSC) file thus skipping the analysis phase.

The UDDI Registry Browser

Pressing the  button in the **WSDL File Opener** dialog (menu **Tools** > **WSDL SOAP Analyzer**) opens the **UDDI Registry Browser** dialog.

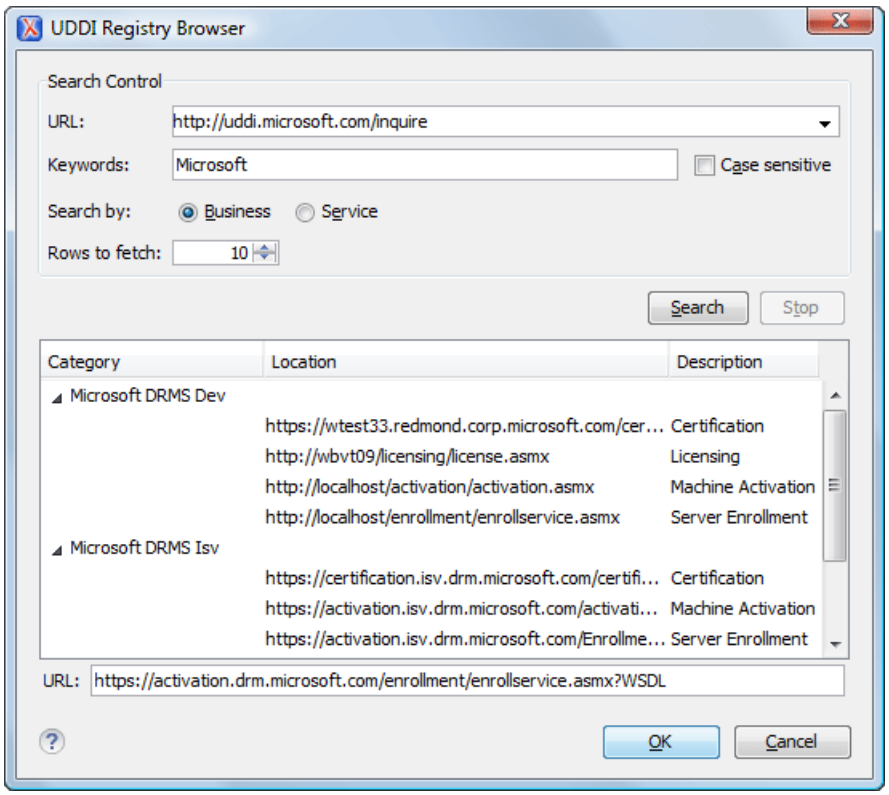


Figure 293: UDDI Registry Browser dialog

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 whether by company or by provided service.
- **Case sensitive** - When checked, the search will take into account the keywords' 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.

Generate WSDL Documentation

To generate documentation for a WSDL document use the action from menu **Tools** > **Generate Documentation** > **WSDL Documentation** .

The WSDL documentation dialog can be also opened from the **Project** tree contextual menu: **Generate Documentation** > **WSDL Documentation...** .

The fields of the dialog are the following:

- **Input URL** - Type the URL of the file or click on the browse button and select it from the file system.
- **Output file (HTML)** - In this field you will have to enter the path and the filename where the documentation will be generated.
- **Open in browser** - If you want the result to be opened in a browser, select this checkbox.

- **Generate** - Press this button for generating the documentation of the WSDL file.

Chapter 20

Digital Signatures

Topics:

- *Overview*
- *Canonicalizing Files*
- *Certificates*
- *Signing Files*
- *Verifying the Signature*

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 allows 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: 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 then 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** and also from menu **Tools** .

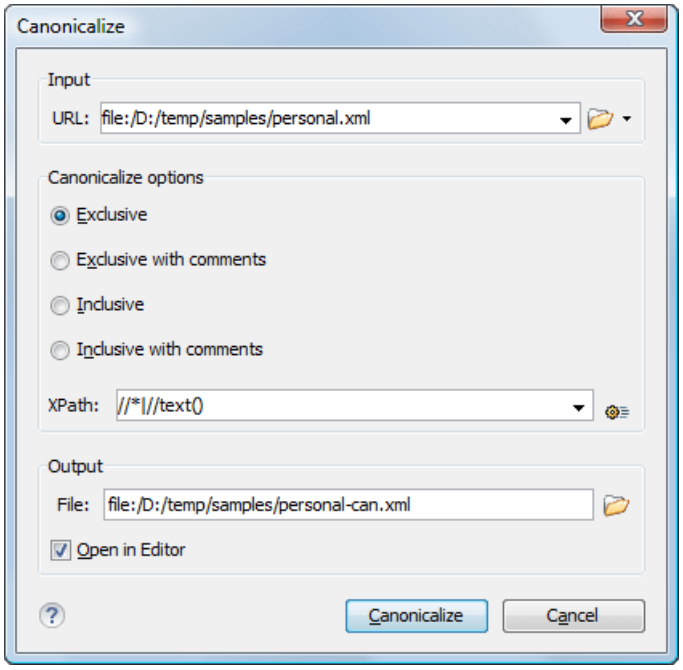


Figure 294: Canonicalization settings dialog

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 it's alias already exists in that keystore and no keystore can store trusted certificates generated with keys in it's keystore.

In Oxygen 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** and also from menu **Tools** .

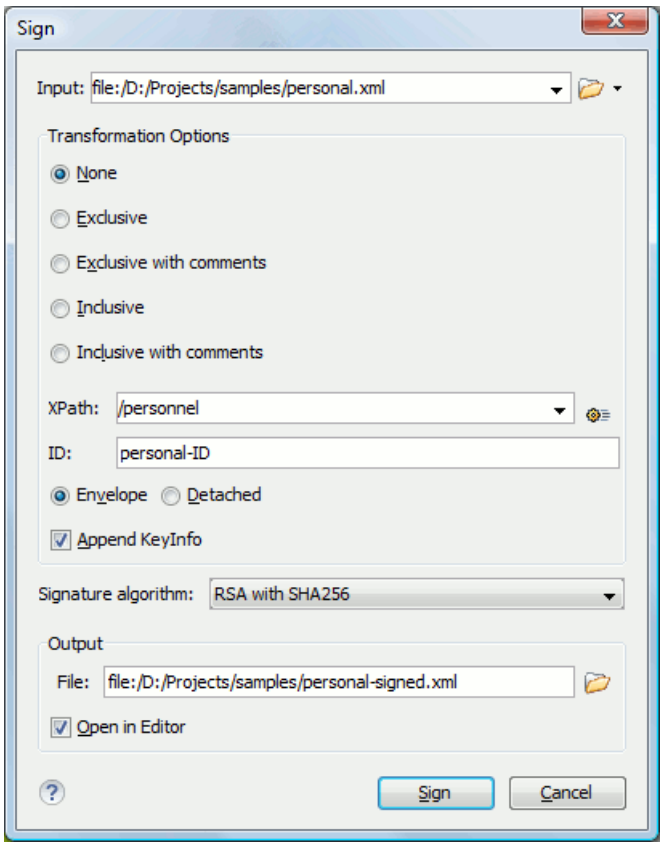


Figure 295: Signature settings dialog

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

The user can select a file to verify its signature in the dialog opened by the action **Verify Signature** available from the editor panel's **contextual menu** > **Source** and also from menu **Tools** . The dialog has a field **URL** that specifies the location of the document for which to verify the signature.

If the signature is valid, a dialog displaying the name of the signer will be opened. If not, an error message will show details about the problem.

Chapter 21

Syncro SVN Client

Topics:

- *Main Window*
- *Getting Started*
- *Syncro SVN Client Views*
- *The Revision Graph of a SVN Resource*
- *Syncro SVN Client Preferences*
- *Command Line Reference*

Syncro SVN is a client for the Subversion version control system compatible with Subversion 1.6 servers. It manages files and directories that change over time and are stored in a central repository. The version control repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to access older versions of your files and examine the history of how and when your data changed.

Main Window

This section explains the main window of Syncro SVN Client.

Views

The main window consists of the following views:

- ***Repositories** view* - Allows you to define and manage Subversion repository locations.
- ***Working Copy** view* - Allows you to manage with ease the content of the working copy.
- ***History** view* - Displays information (author name, revision number, commit message) about the changes made to a resource during a specified period of time.
- ***Editor** view* - Allows you to edit different types of text files, with full syntax-highlight.
- ***Annotations** view* - Displays a list with information regarding the structure of a document (author and revision for each line of text).
- ***Compare** view* - Displays the differences between two revisions of a text file from the working copy.
- ***Image Preview*** - Allows you to preview standard image files supported by Syncro SVN Client: JPG, GIF and PNG.
- ***Compare Images** view* - Displays two images side by side.
- ***Properties** view* - Displays the SVN properties of a resource under version control.
- ***Console** view* - Displays information about the currently running operation, similar with the output of the Subversion command line client.
- ***Help** view* - Shows information about the currently selected view.

The main window's status bar presents in the left side the operation in progress or the final result of the last performed action. In the right side there is a progress bar for the running operation and a stop button to cancel the operation.

Main Menu

The main menu of the Syncro SVN Client is composed of the following menus:

- **File** menu:
  - **New** submenu:
    - **New File** - This operation creates a file in the working copy and adds it to version control. If the selected path is not under version control, the newly created file is added to the repository only by an explicit action. Creating a file in the working copy does not add it automatically to the repository. This action works only for selected paths in the **Working Copy** tree.
    - **New Folder (Ctrl + Shift + F)** - This operation creates a new folder as child of the selected folder from the ***Repositories** view* tree or from the ***Working Copy** view* tree, depending on which view was focused last when performing this action. For the ***Working Copy** view*, the folder is added to version control only if the selected path is under version control, otherwise the newly created directory is not added to version control.
    - **New External Folder (Ctrl + Shift + W)** - This operation sets a folder name in the property `svn:externals` of the selected folder. The repository URL to the folder to which the new external folder points and the revision number of that repository URL can be selected easily with the **Browse** and **History** buttons of the dialog. This action works only for selected paths in the ***Working Copy*** tree.

      Subversion clients 1.5 and higher support relative external URLs. You can specify the repository URLs to which the external folders point using the following relative formats:

      - **../** - Relative to the URL of the directory on which the `svn:externals` property is set.
      - ^/ - Relative to the root of the repository in which the `svn:externals` property is versioned.
      - // - Relative to the scheme of the URL of the directory on which the `svn:externals` property is set.
      - / - Relative to the root URL of the server on which the `svn:externals` property is versioned.

- **Open (Ctrl + O)** - This action opens the selected file in an editor where you can modify it. The action is active only when a single item is selected. The action opens a file with the internal editor or the external application associated with that file type. In case of a folder the action opens the selected folder with the system application for folders (for example Windows Explorer on Windows, Finder on Mac OS X, etc). Folder opening is available only for folders selected in the ***Working Copy** view*. This action works on any file selection from the ***Repositories** view*, ***Working Copy** view*, ***History** view* or ***Directory Change Set** view*, depending on which view was last focused when invoking it.
- **Open with ...(Ctrl + Shift + O)** - Displays the ***Open with*** dialog for specifying the editor in which the selected file is opened. In case multiple files are selected only external applications can be used to open the files. This action works on any file selection from ***Repositories** view*, ***Working Copy** view*, ***History** view* or ***Directory Change Set** view*, depending on which view was last focused when invoking it.
- **Save (Ctrl + S)** - Saves the local file currently opened in the editor or the **Compare** view.
- **Copy URL Location (Ctrl + Alt + U)** - Copies to clipboard the URL location of the resource currently selected in the **Repositories** view.
- **Copy/Move to (Ctrl + M)** - Copies or moves to a specified location the resource currently selected either in **Repositories** or **Working copy** view.
- **Rename (F2)** - Renames the resource currently selected either in **Repositories** or **Working copy** view.
- **Delete (Delete)** - Deletes the resource currently selected either in **Repositories** or **Working copy** view.
- **Show SVN Properties (Ctrl + Shift + P)** - Brings up the ***Properties** view* and displays the SVN properties for a selected resource from ***Repositories** view* or ***Working Copy** view*, depending on which view was last focused when invoking it.
- **File Information (Ctrl + I)** - Provides additional information for a selected resource from the ***Working Copy** view*. For more details please see the section *Obtain information for a resource*.
- **Exit (Ctrl + Q)** - Closes the application.

- **Edit** menu:
  - **Undo (Ctrl + Z)** - Undo edit changes in the local file currently opened in the editor or the **Compare** view.
  - **Redo (Ctrl + Y)** - Redo edit changes in the local file currently opened in the editor or the **Compare** view.
  - **Cut (Ctrl + X)** - Cut selection to clipboard from the local file currently opened in the editor view or the **Compare** view.
  - **Copy (Ctrl + C)** - Copy selection to clipboard from the local file currently opened in the editor or the **Compare** view.
  - **Paste (Ctrl + V)** - Paste selection from clipboard in the local file currently opened in editor or the **Compare** view.
  - **Find/Replace (Ctrl + F)** - Perform find / replace operations in the local file currently opened in the editor or the **Compare** view.
  - **Find Next (F3)** - Go to the next find match using the same find options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.
  - **Find Previous (Shift + F3)** - Go to the previous find match using the same find options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.
- **Repository** menu:
  - **New Repository Location (Ctrl + Alt + N)** - Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.

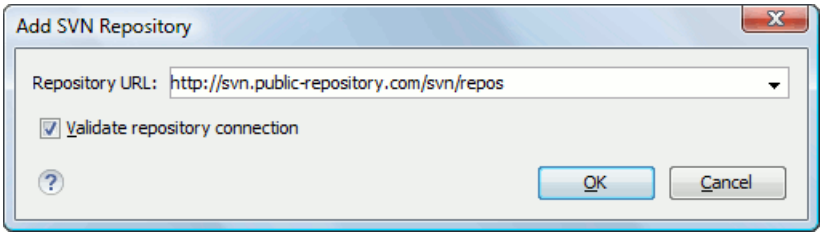


Figure 296: Add SVN Repository

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

- **Edit Repository Location (Ctrl + Alt + E)** - Context-dependent action that allows you to edit the selected repository location by means of the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.
- **Change the Revision to Browse (Ctrl + Alt + Shift + B)** - Context-dependent action that allows you to change the selected repository revision by means of the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.
- **Remove Repository Location (Ctrl + Alt + Shift + R)** - Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.
- **Refresh** - Refreshes the resource selected in the **Repositories** view.
- **Check Out (Ctrl + Alt + Shift + C)** - Allows you to copy resources from a repository into your local file system. To use this operation, you must select a repository root location or a folder from a repository, but never a file. If you don't select anything, you can specify an URL to a folder resource from a repository in the **Check Out** dialog that appears when performing this operation. To read more about this operation, see the section *Check out a working copy*.
- **Export** - Exports a folder from the repository to the local file system.
- **Import** sub-menu:
  - **Import Folder Content (Ctrl + Alt + Shift + M)** - Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.
  - **Import File(s) (Ctrl + Alt + I)** - Imports the files selected from the files system into the selected folder from the repository.

- **Working Copy** menu:
  - **Add / Remove Working Copy** - Opens dialog with a list of working copies that the Subversion client is aware of. In this dialog you can add existing working copies or remove no longer needed ones.
  - **Switch to** - Selects one of the following view modes: **All Files**, **Modified**, **Incoming**, **Outgoing**, or **Conflicts**.
  - **Refresh (F5)** - Refreshes the state of the selected resources or of the entire working copy if there is no selection.
  - **Synchronize (Ctrl + Shift + S)** - Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.
  - **Update (Ctrl + U)** - Updates all the selected resources that have incoming changes to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.
  - **Update to revision/depth** - Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the *sparse checkouts* section.

- **Commit** - Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.
- **Update all (Ctrl + Shift + U)** - Updates all resources from the working copy that have incoming changes. It performs a recursive update on the synchronized resources.
- **Commit all** - Commits all the resources with outgoing changes. It is disabled when **Incoming** mode is selected or the synchronization result does not contain resources with outgoing changes. It performs a recursive commit on the synchronized resources.
- **Revert (Ctrl + Shift + V)** - Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Subversion's pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.
- **Edit conflict (Ctrl + E)** - Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see *Edit conflicts*.
- **Mark Resolved (Ctrl + Shift + R)** - Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.
- **Mark as Merged (Ctrl + Shift + M)** - Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.
- **Override and Update** - Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.
- **Override and Commit** - Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.
- **Add to version control (Ctrl + Alt + V)** - Adds the selected resources to version control. A directory will be added recursively to version control. It is not mandatory to explicitly add resources to version control but it is recommended. At commit time unversioned resources will have to be manually selected in the commit dialog. This action is only active on unversioned resources.
- **Add to "svn:ignore" (Ctrl + Alt + I)** - Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the *svn:ignore* property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.
- **Cleanup (Ctrl + Shift + C)** - Performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.
- **Locking**:
  - **Scan for locks (Ctrl + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see *Scanning for locks*.
  - **Lock (Ctrl + K)** - Allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force(*steal*) the lock. The action is active only on files under version control. For more details on the use of this action see *Locking a file*.
  - **Unlock (Ctrl + Alt + K)** - Releases(unlocks) the exclusive access to a file from the repository. You can also choose to unlock it by force(*break the lock*).
- **Expand all (Ctrl + Alt + X)** - Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.

- **Collapse all (Ctrl + Alt + Z)** - Collapses all descendants of the selected folder. The same behavior is obtained by double-clicking on a expanded folder.

- **Compare** menu:
  - **Perform Files Differencing (Ctrl + D)** - Performs file differencing on request.
  - **Go to First Modification (Ctrl + B)** - Navigates to the first difference.
  - **Go to Previous Modification (Ctrl + Shift + N)** - Navigates to the previous difference.
  - **Go to Next Modification (Ctrl + N)** - Navigates to the next difference.
  - **Go to Last Modification (Ctrl + E)** - Navigates to the last difference.
  - **Copy All Non-Conflicting Changes from Right to Left** - This action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.
  - **Copy Change from Right to Left (Ctrl + Shift + Comma)** - This action copies the selected change from the right editor to the left editor.
  - **Show Modification Details at Word Level** - Because the differences are computed using a line differencing algorithm sometimes is useful to see exactly what words are different in a changed section.
  - **Show Modification Details at Character Level** - Useful when you want to find out exactly what characters are different between the two analyzed sections.
  - **Ignore Whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before the strings are compared they are first normalized and then the whitespace at the beginning and the end of the strings is trimmed.

- **History** menu:
  - **Show History (Ctrl + H)** - Displays the history for a SVN resource at a given revision. The resource can be one selected from the **Repositories** view, **Working Copy** view, or from the **Affected Paths** table from the **History** view, depending on which view was last focused when this action was invoked.
  - **Show Annotation (Ctrl + Shift + A)** - Complex action that does the following operations:
    - opens the selected resource in the **Annotations** editor;
    - displays corresponding annotations list in the **Annotations** view;
    - displays the history of the selected resource.
  - This operation is available for any resource selected from **Repositories** view, **Working Copy** view, **History** view or **Directory Change Sets** view, depending on which view was last focused when this action was invoked.
  - **Revision Graph (Ctrl + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

- **Tools** menu:
  - **Branch / Tag** - Allows you to copy the selected resource from the **Repositories** view or **Working Copy** view to a branch or tag into the repository. To read more about this operation, see the section *Creating a Branch / Tag*.
  - **Merge (Ctrl + J)** - Allows you to merge the changes made on one branch back into the trunk, or vice versa, using the selected resource from the working copy. To read more about this operation, see the section *Merging*.
  - **Switch (Ctrl + Alt + W)** - Allows you to change the repository location of a working copy or only of a versioned item of the working copy within the same repository. It is available when the selected item of the working copy is a versioned resource, except an external folder. To read more about this action, see the section *Switching the Repository Location*.
  - **Relocate** - Allows you to change the base URL of the root folder of the working copy to a new URL, when the base URL of the repository changed, for example the repository itself was relocated to a different server. This operation is available for a selected item of the working copy tree that is a versioned folder. To read more about this operation, see the section *Relocate a Working Copy*.

- **Create patch (Ctrl + Alt + P)** - Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see *the section about patches*.
- **Working copy format** - this submenu contains the following two operations:
  - **Upgrade** - Allows you to upgrade the format of the current working copy to the newest one known by Syncro SVN Client, to allow you to benefit of all the new features of the client.
  - **Downgrade** - Allows you to downgrade the format of the current working copy to an older format. The formats allowed to downgrade to are SVN 1.5 and SVN 1.4. This is useful in case you wish to use older SVN clients with the current working copy, or, by mistake, you have upgraded the format of an older working copy by using a newer SVN client.

  See the section *Working Copy Format* to read more about this subject.

- **Options** menu:
  - **Preferences** - Opens the **Preferences** dialog.
  - **Menu Shortcut Keys** - Opens the **Preferences** dialog directly on the **Menu Shortcut Keys** option page, where users can configure in one place the keyboard shortcuts available for menu items available in Syncro SVN Client.
  - **Global Run-Time Configuration** - Allows you to configure SVN general options, that should be used by all the SVN clients you may use:
    - **Edit 'config' file** - In this file you can configure various SVN client-side behaviors.
    - **Edit 'servers' file** - In this file you can configure various server-specific protocol parameters, including HTTP proxy information and HTTP timeout settings.
  - **Export Options** - Allows you to export the current options to a file.
  - **Import Options** - Allows you to import options you have previously exported.
  - **Reset Options** - Resets all your options to the default ones.
  - **Reset Authentication** - Resets the Subversion authentication information.
- **Window** menu:
  - **Show View** - Allows you to select the view you want to bring to front.
  - **Show Toolbar** - Allows you to select the toolbar you want to be visible.
  - **Reset Layout** - Resets all the views to their default position.
- **Help** menu:
  - **Help (F1)** - Opens the **Help** dialog.
  - **Dynamic Help** - Shows the **Dynamic Help** view.
  - **Check for New Versions** - Checks the availability of new Syncro SVN Client versions.
  - **Register** - Opens the registration dialog.
  - **Improvement Program Options** - Allows you to activate or deactivate the Syncro Soft Product Improvement Program.
  - **Report Problem** - Opens a dialog that allows the user to write the description of a problem that was encountered while using the application.
  - **Support Center** - Opens the Support Center web page in a browser.

Main Toolbar

The toolbar of the SVN Client SVN Repositories window contains the following actions:

- **Check Out** - Checks out a working copy from a repository. The repository URL and the working copy format must be specified.
- **Synchronize** - Synchronizes the current working copy with the repository.

- **Update All** - Updates all resources of the working copy that have an older revision that repository.
- **Commit All** - Commits all resources of working copy that have a newer version compared to that of the repository.
- **Refresh** - Refreshes the whole content of the current working copy from disk starting from the root folder. At the end of the operation, the modified files and folders that were not committed to repository yet, are displayed in the **Working Copy** view.
- **Compare** - The selected resource is compared with:
  - the *BASE* revision, when the selected resource is:
    - locally modified and the **All Files** view mode is currently selected (no matter if there are incoming changes);
    - locally modified and there are no incoming changes when any other view mode is selected.
  - the remote version of the same resource, when remote information is available after a **Synchronize** operation (only when one of **Modified**, **Incoming**, **Outgoing** and **Conflicts** view modes is selected).
  - the working copy revision, when the selected resource is from the **History** view;
- **Show History** - Displays the history of the selected resource (from the **Working Copy** or **Repository** views) in the **History** view.
- **Show Annotation** - Displays the annotations of the selected resource. The selected resource can be in the **Working Copy** or the **History** views.
- **Revision Graph** - Displays the revision graph of the selected resource. The selected resource can be in the **Working Copy** or the **Repositories** views.

Status Bar

The status bar of the Syncro SVN Client window displays important details of the current status of the application. This information is available only in the **Working Copy** view.

Figure 297: Status bar

The status bar is composed of the following areas:

- the path of the currently processed file from the current working copy (during an operation like **Checkout** or **Synchronize**) or the result of the last operation;
- the current status of the following working copy options:
  - Show ignored files ( )
  - Show deleted files ( )
  - Process `svn:externals` definitions ( )

  The options for ignored and deleted files are switched on and off from *the **Settings** menu* of the **Working Copy** panel;
- the current numbers of incoming changes ( ), outgoing changes ( ) and conflicting changes ( );
- a progress bar for the currently running SVN operation and a button ( ) that allows you to stop it.

Getting Started

This section explains the basic operations that can be done in Syncro SVN Client.

SVN Repository Location

This section explains how to add and edit the repository locations in Syncro SVN Client.

Add / Edit / Remove Repository Locations

Usually team members do all of their work separately, in their own working copies and must share their work. This is done via a Subversion repository. Syncro SVN Client supports the versions 1.3, 1.4, 1.5 and 1.6 of the SVN repository format.

Before you can begin working with a Subversion repository, define a repository location in the ***Repositories** view*.

To create a repository location, click the **New Repository Location** toolbar button or right click inside the view and select **New Repository Location...** from the popup menu. On Windows, the context menu can be displayed on a right click with the mouse or with the keyboard by pressing the special context menu key available on Windows keyboards. This action opens the **Add SVN Repository** dialog which prompts you for the URL of the repository you want to connect to. No authentication information is requested at the time the location is defined. It is left to the Subversion client to request the user and password information when it is needed. The main benefit of allowing Subversion to manage your password in this way is that it prompts you for a new password only when your password changes.

Once you enter the repository URL Syncro SVN Client tries to contact the server and get the content of the repository for displaying it in the ***Repositories** view*. If the server does not respond in the timeout interval *set in Preferences*, an error is reported. If you do not want to wait until the timeout expires, you can end the waiting process with the **Stop** button from the toolbar of the view.

To edit a repository location, click the **Edit Repository Location** toolbar button or right click inside the view on a repository root entry and select **Edit Repository Location...** from the popup menu.

The **Edit SVN Repository** dialog works in the same way as the **Add SVN Repository** dialog. It shows the previously defined repositories URLs and it allows you to change them.

To remove a repository location, click the **Remove Repository Location** toolbar button or right click inside the view on a repository entry and select **Remove Repository Location...** from the popup menu. A confirmation dialog is displayed to make sure that you do not accidentally remove locations.

The order of the repositories can be changed in the **Repositories** view at any time with the two buttons on the toolbar of the view, the up arrow and the down arrow. For example, pressing the up arrow once moves up the selected repository in the list with one position.

To set the reference revision number of an SVN repository right-click on the repository in the list displayed in the ***Repositories** view* and select the **Change the Revision to Browse...** action. The revision number of the repository is used for displaying the contents of the repository when it is viewed in the ***Repositories** view*. Only the files and folders that were present in the repository at the moment when this revision number was generated on the repository are displayed as contents of the repository tree. Also this revision number is used for all the file open operations executed directly from the ***Repositories** view*.

Authentication

Five protocols are supported: *HTTP*, *SVN*, *HTTPS*, *SVN + SSH* and *FILE*. If the repository that you are trying to access is password protected, the **Enter authentication data** dialog requests a user name and a password. If the **Store authentication data** checkbox is checked, the credentials are stored in Subversion's default directory:

- on Windows - `%HOME%\Application Data\Subversion\auth`. Example: `C:\Documents and Settings\John\Application Data\Subversion\auth`

- on Linux and Mac OS X - `$HOME/.subversion/auth`. Example: `/home/John/.subversion/auth`

There is one file for each server that you access. If you want to make Subversion forget your credentials, you can use the **Reset authentication** command from the **Options** menu. This causes Subversion to forget all your credentials.

Note: When you reset the authentication data, restart the application in order for the change to take effect.

Tip: The FILE protocol is recommended if the SVN server and Syncro SVN Client are located on the same computer as it ensures faster access to the SVN server than the other protocols.

For HTTPS connections where client authentication is required by your SSL server, you must choose the certificate file and enter the corresponding certificate password which is used to protect your certificate.

When using a secure HTTP (HTTPS) protocol for accessing a repository, a **Certificate Information** dialog pops up and asks you whether you accept the certificate permanently, temporarily or simply deny it.

If the repository has SVN+SSH protocol, the SSH authentication can also be made with a private key and a pass phrase.

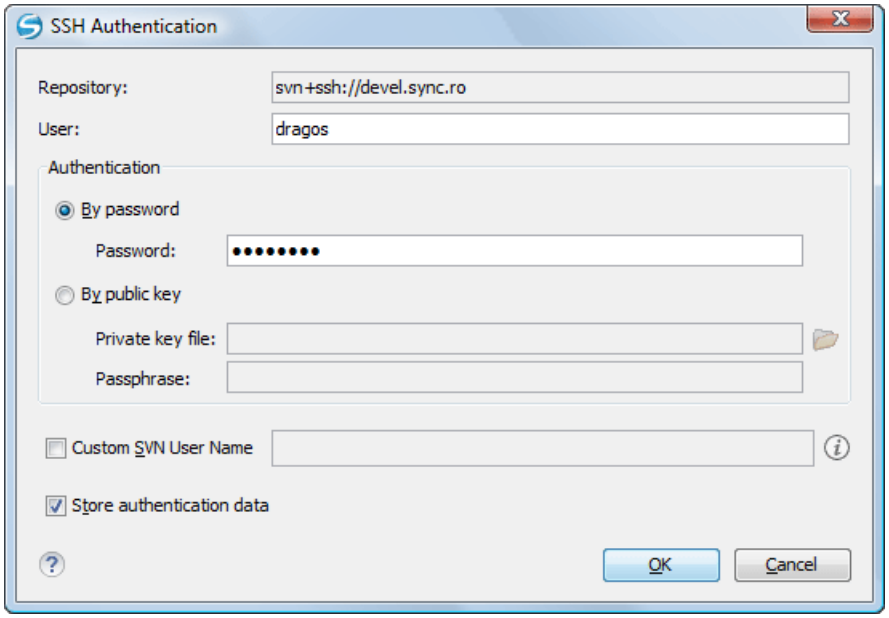


Figure 298: User & Private key authentication dialog

After the SSH authentication dialog, another dialog pops up for entering the SVN user name that accesses the SVN repository. The SVN user name is recorded as the committer in SVN operations.

Defining a Working Copy

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, your working copy being your private work area. In order to make your own changes available to others or incorporate other people's changes, you must explicitly tell Subversion to do so. You can even have multiple working copies of the same project.

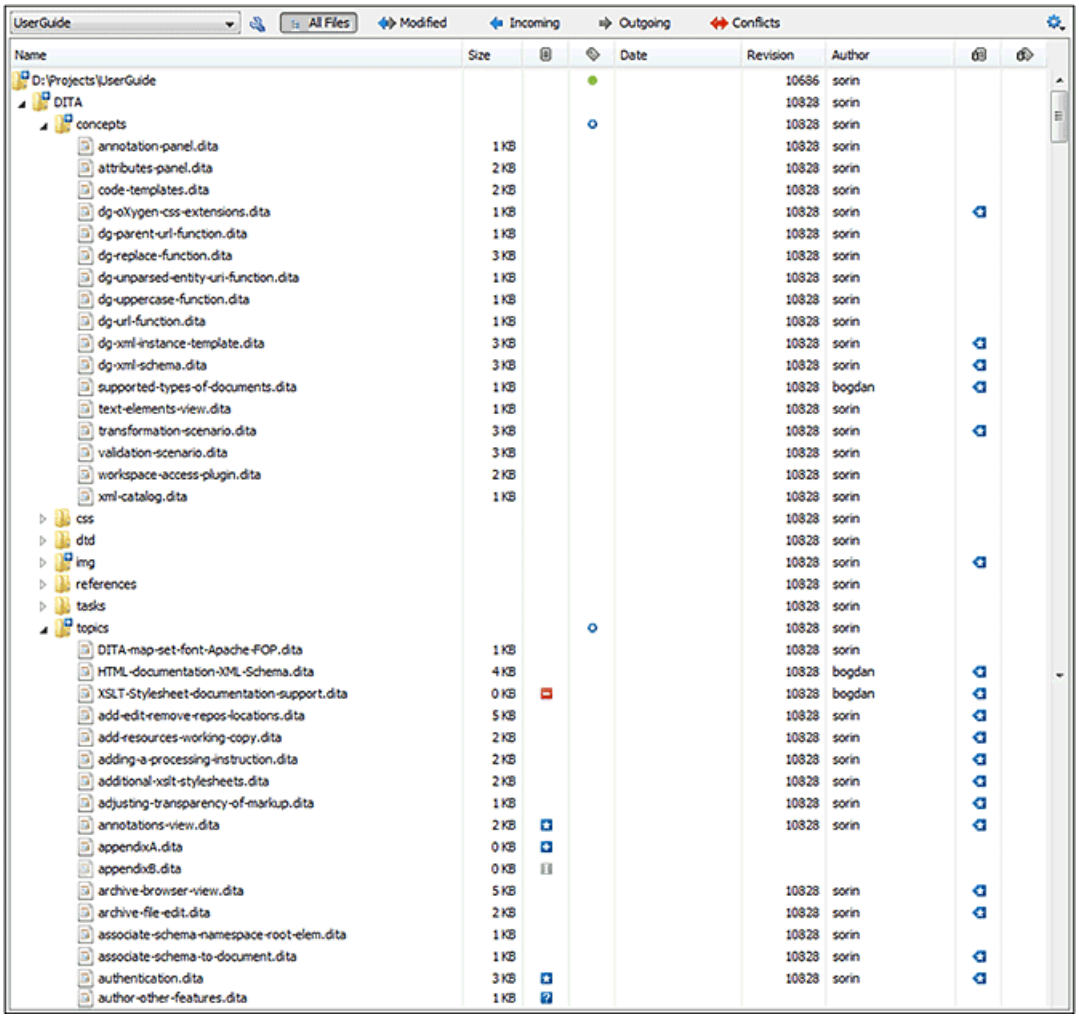


Figure 299: Working Copy View

A Subversion working copy also contains some extra files, created and maintained by Subversion, to help it keep track of your files. In particular, each directory in your working copy contains a subdirectory named `.svn`, also known as the working copy *administrative directory*. This administrative directory contains an unaltered copy of the last updated files from the repository. This copy is usually referred to as the *pristine copy* or the *BASE revision* of the working copy. These files help Subversion recognize which files contain unpublished changes, and which files are out-of-date with respect to others' work.

A typical Subversion repository often holds the files (or source code) for several projects. Usually each project is a subdirectory in the repository's file system tree. In this arrangement, a user's working copy usually corresponds to a particular subtree of the repository.

Check Out a Working Copy

Check Out is the term used to describe the process of making a copy of a project from a repository into your local file system. This checked-out copy is called a working copy. A Subversion working copy is a specially formatted folder structure which contains additional `.svn` folders that store Subversion information, as well as a pristine copy of each item that is checked out.

You check out a working copy from the ***Repositories** view*. If you have not yet defined a connection to your repository, you need to *add a new repository location*.

1. Navigate to the desired repository folder in the **Repositories** view.
2. Right click on the folder and select **Check Out...** from the popup menu.

The **Check Out** dialog is displayed:

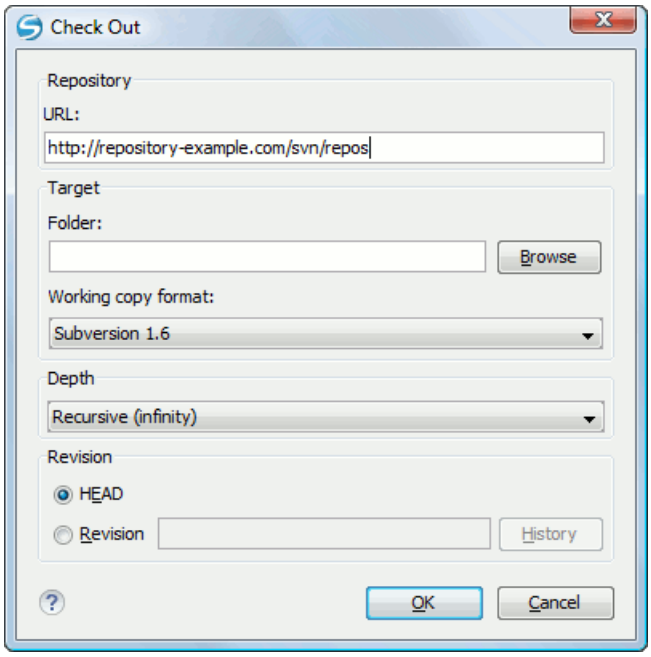


Figure 300: Check Out Dialog

3. Click on the **Browse** button.
4. Select the location where the working copy is created.
5. Select the version of the working copy format: SVN 1.4, SVN 1.5 or SVN 1.6.
6. Select the depth for the checkout folder in the **Depth** combo box.

   This allows you to specify the recursion level into child resources. It is used if you want to check out only a portion of a working copy and then bring in a future update operation previously ignored files and subdirectories. You can find out more about checkout depth in the *sparse checkouts* section.
7. Select the revision number that is checked out.

   By default the last (HEAD) revision is checked out. If you need another revision, you have to select the **Revision** radio button. To specify the revision number you can simply type the revision number in the corresponding text field or click on the **History** button which opens *the **History** dialog*.

After a check out operation, the new working copy will be added to the list in the ***Working Copy** view*. The working copy content is displayed in that view.

The History Dialog

The **History** dialog presents a list of revisions for a resource. It is opened from the dialogs that require setting an SVN revision number like *the **Check Out** dialog* or *the **Branch / Tag** dialog*. It presents information about revision, commit date, author, and commit comment.

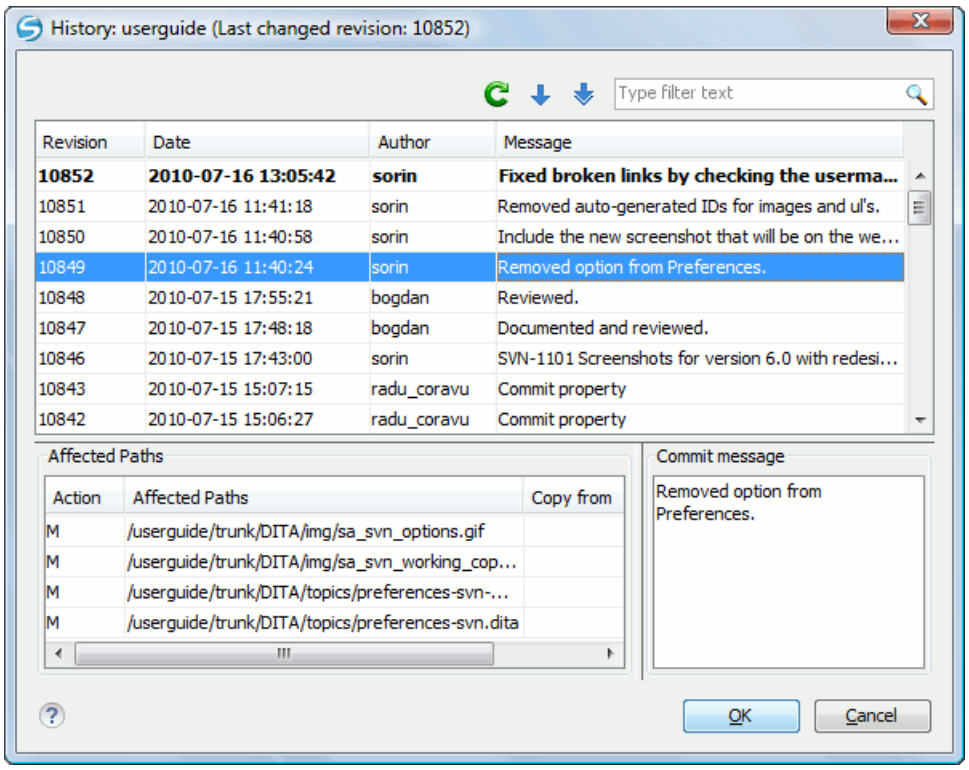


Figure 301: History Dialog

The initial number of entries in the list is 50. Additional revisions can be added to the list using the **Get next 50** and **Get all** buttons. The list of revisions can be refreshed at any time with the **Refresh** button.

The **Affected Paths** area displays all paths affected by the commit of the revision selected in history. The contextual menu invoked on a revision selected in the **Affected Paths** area contains the following actions:

- **Compare with previous version** - Makes a diff between the selected revision and the previous one. If there is no external application specified for executing diff operations, the built-in diff tool is applied. The same action is also executed when double clicking a file in the **Affected Paths** area.
- **Open** - Opens the revision in the editor panel.
- **Save revision to ...** - Saves the revision to a new file.
- **Revert changes from this revision** - The changes committed by the selected revision are reverted in the current version of the file in the working copy. If the committed changes represented in fact an SVN delete operation, the result is restoring the deleted file in the working copy.
- **Update to revision** - Makes the selected revision the current revision in the working copy.
- **Show History** - Displays the history of the selected revision.
- **Show Annotation** - Opens the **Annotations** view for the selected revision.

Use an Existing Working Copy

Using an existing working copy is the process of taking a working copy that exists on your file system and connecting it to Subversion. If you have a brand new project that you want to import into your repository, then see the section *Import resources into the repository*. The following procedure assumes that you have an existing valid working copy on your file system.

1. Click on the **Add / Remove Working Copy** toolbar button in the ***Working Copy** view*.
   This action opens the **Working copies list** dialog.
2. Press the **Add** button.
3. Select the working folder copy from the file system.

4. Optionally you can press the **Edit** button to change the name of the working copy that is displayed in the **Working Copy** view.

   The name is useful to differentiate between working copies located in folders with the same name. The default name is the name of the root folder of the working copy.

   The order of the working copies can be changed in the list using the two arrow buttons which move the selected working copy with one position up or down.

5. Press the **OK** button.

The selected working copy is loaded and presented in the ***Working Copy** view*.

Manage Working Copy Resources

This section explains how to work with the resources that are displayed in the **Working Copy** view.

Edit Files

You can edit files from the ***Working Copy** view* by double clicking them or by right clicking them and choosing **Open** from the contextual menu.

Please note that only one file can be edited at a time. If you try to open another file, it is opened in the same editor window. The editor has syntax highlighting for known file types, meaning that a different color is used for each type of recognized token in the file. If the selected file is an image, then it is previewed in the editor, with no access to modifying it.

After modifying and saving a file from a working copy, a modified marker - an asterisk (*) - will be added to the file's icon in the ***Working Copy** view*. The asterisk marks the files that have local modifications that were not committed to the repository.

Add Resources to Version Control

The new files and folders you create during the development process must be added to Version Control, using the **Add** command from the contextual menu in the ***Working Copy** view*. If you do not do this, the resource is marked with a question mark (?), meaning that it is *unversioned* (unknown). After you have added it to version control, the resource will be marked as *added* (+), which means you first have to commit your working copy to make that resource available to other developers. Adding a resource to version control does not affect the repository.

If you try to add to version control an unversioned directory, the entire subtree starting with that directory is added.

When you *commit your changes*, if you forgot to add a resource, it will still be presented in the commit dialog, but will be de-selected by default. When you commit the unversioned resource, it is automatically added to version control before being committed and the marking is removed.

Ignore Resources Not Under Version Control

Some resources inside your working copy do not need to be subject to version control. These resources can be files created by the compiler, `*.obj`, `*.class`, `*.lst`, or output folders used to store temporary files. Whenever you *commit changes*, Subversion shows your modified files but also the unversioned files, which fill up the file list in the commit dialog. Though the unversioned files are committed unless otherwise specified, it is difficult to see exactly what you are committing.

The best way to avoid these problems is to add the derived files to the Subversion's ignore list. That way they are never displayed in the commit dialog and only genuine unversioned files which must be committed are shown.

You can choose to ignore a resource by using the **Add to svn:ignore** action in the contextual menu of the ***Working Copy** view*.

In the **Add to svn:ignore** dialog you can specify the resource to be ignored by name or by a custom pattern. The custom pattern can contain the following wildcard characters:

- * - Matches any string of characters of any size, including the empty string.

- ? - Matches any single character.

For example, you can choose to ignore all text documents by using the pattern: **.txt*.

The action **Add to svn:ignore** adds a predefined Subversion property called `svn:ignore` to the parent directory of the specified resource. In this property, there are specified all the child resources of that directory that must be ignored. The result is visible in the **Working Copy** view. The ignored resources are represented with grayed icons.

Delete Resources

The delete command can be found in the **Edit** submenu of the context menu from the ***Working Copy** view*. When you delete a resource from the Subversion working copy, it is removed from the file system and it is also marked as deleted. If unversioned, added or modified resources are encountered, a dialog prompts you to confirm their deletion.

The delete command does not delete from the file system the directories under version control, it only marks them as deleted. This is because the directories also contain the pristine copy of that directory content. In the ***Working Copy** view* this action is transparent as all resources have the deleted mark (the minus (-) sign). The directories are removed from the file system when you *commit* them to the repository. You can also change your mind completely and *revert* the deleted files to their initial, pristine state.

If you delete a resource from the file system without Subversion's knowledge, you render the working copy in an inconsistent state.

If a resource is deleted from the file system without Subversion's knowledge, your working copy is in an inconsistent state. The resource will be considered and marked as missing (the sign '!'). If a file was deleted, it will be treated in the same way as if it was deleted by Subversion. However if a directory is missing you will be unable to commit. If you *update your working copy*, Subversion will replace the missing directory with the latest version from the repository and you can then delete it the correct way using the **Delete** command.

👉 **Note:** The **Delete** action is not enabled when the selection contains *missing* resources.

Copy Resources

You can copy several resources from different locations of the working copy. You select them in the ***Working Copy** view* and then you initiate the copy command from the contextual menu. This is not a simple file system copy but a Subversion command. It will copy the resource and the copy will also have the original resource's history. This is one of Subversion's very important features, as you can keep track of where the copied resources originated.

Please note that you can only copy resources that are under version control and are committed to the repository or unversioned resources. You cannot copy resources that are added but not yet committed.

In the **Copy File(s)** dialog you can navigate through the working copy directories in order to choose a target directory. If you try to copy a single resource you are also able to change that resource's name in the corresponding text field.

If an entire directory is copied the **Override and Update** action will be enabled only for it and not for its descendants. In the **Commit** dialog will appear only the directory in question without its children.

Move Resources

As in the case of the copy command you can perform the move operation on several resources at once. Just select the resources in the ***Working Copy** view* and choose the **Move** command from the contextual menu. The move command actually behaves as if a copy followed by a delete command were issued. You will find the moved resources at the desired destination and also at their original location but marked as deleted.

Rename Resources

The rename action can be found in the contextual menu of the ***Working Copy** view*. This action can only be performed on a single resource. The rename command acts as a move command with the destination being the same as the original location of the resource. A copy of the original resource will be made with the new name and the original will be marked as deleted.

Lock / Unlock Resources

The idea of version control is based on the copy-modify-merge model of file sharing. This model states that each user contacts the repository and creates a local working copy (check out). Users can then work independently and make modifications to their working copies as they please. When their goal has been accomplished, it is time for the users to share their work with the others, to send them to the repository (commit). When a user has modified a file that has been also modified on the repository, the two files will have to be merged. The version control system assists the user with the merging as much as it can, but in the end the user is the one that must make sure it is done correctly.

The copy-modify-merge model only works when files are contextually mergeable: this is usually the case of line-based text files (such as source code). However this is not always possible with binary formats, such as images or sounds. In these situations, the users must each have exclusive access to the file, ending up with a lock-modify-unlock model. Without this, one or more users could end up wasting time on changes that cannot be merged.

A Subversion lock is a piece of metadata which grants exclusive access to a user. This user is called the lock owner. A lock is uniquely identified by a lock token (a string of characters). If someone else attempts to commit the file (or delete a parent of the file), the repository will demand two pieces of information:

- User authentication - The user performing the commit must be the lock owner.
- Software authorization - The user's working copy must have the same lock token as the one from the repository, proving that it is the same working copy where the lock originated from.

Scanning for Locks

When starting to work on a file that is not contextually mergeable (usually a binary file), it is better to verify if someone else isn't already working on that file. You can do this in the ***Working Copy** view* by selecting one or more resources, then right clicking on them and choosing the **Scan for Locks** action from the context menu.

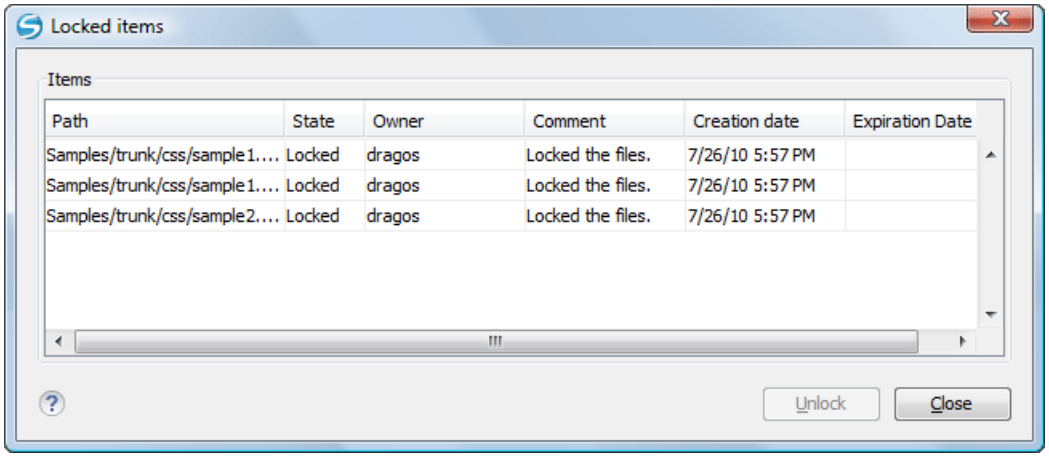


Figure 302: The locked items dialog

The **Locked items** dialog contains a table with all the resources that were found locked on the repository. For each resource there are specified: resource path, state of the lock, owner of the lock, lock comment, creation and expiration date for the lock (if any).

The state of the lock can be one of:

- **Other** - If someone else locked the file.
- **Locked** - If the current user locked the file.
- **Broken** - If the current user locked the file but it was forcefully unlocked by someone else afterwards.
- **Stolen** - If the current user locked the file but it was forcefully locked by someone else afterwards.

You can unlock a resource by selecting it and pressing the **Unlock** button.

Locking a File

A locked file allows you exclusive write access to a file from the repository, meaning that you are the only one who can modify and commit the file to the repository.

You can lock a file from the contextual menu of the ***Working Copy** view*. Note that you can only lock several files at once but no directories. This is a restriction of Subversion which is used to discourage the use of the lock-modify-unlock model at large scale or when unnecessary.

In the **Lock** dialog you can write a comment for the lock and if necessary steal (force) the lock. Note that you should only steal a lock after you made sure that the previous owner no longer needs it, otherwise you may cause an unsolvable conflict which is exactly why the lock was put there in the first place. The Subversion server can have a policy concerning lock stealing, it may not allow you to steal a lock if a certain condition is not satisfied.

The lock will stay in place until you *commit* the locked file or until someone unlocks it. There is also the possibility that the lock will expire after a period of time specified in the Subversion server policy.

Unlocking a File

A file can be unlocked from the contextual menu of the ***Working Copy** view*. A dialog will prompt you to confirm the unlocking and it will also allow you to break the lock (unlock it by force).

Synchronize with Repository

In the work cycle you will need to incorporate other people's changes (update) and to make your own work available to others (commit). This is what the **Incoming** and **Outgoing** modes of *the **Working Copy** view* was designed for, to help you send and receive modifications from the repository.

The **Incoming** and **Outgoing** modes of this view focus on incoming and outgoing changes. The incoming changes are the changes that other users have committed since you last updated your working copy. The outgoing changes are the modifications you made to your working copy as a result of editing, removing or adding resources.

The view presents the status of the working copy resources against the BASE revision after a **Refresh** operation. You can view the state of the resources versus a repository HEAD revision by using the **Synchronize** action from *the **Working Copy** view*.

View Differences

One of the most common requirements in project development is to see what changes have been made to the files from your Working Copy or to the files from the repository. You can examine these changes after a synchronize operation with the repository, by using the **Open in compare editor** action from the contextual menu.

The text files are compared using a built-in ***Compare** view* which uses a line differencing algorithm or a specified external diff application if such an application is *set in the SVN preferences*. When a file with outgoing status is involved, the compare is performed between the file from the working copy and the BASE revision of the file. When a file with incoming or conflict status is involved, the differences are computed using a three-way algorithm which means that the local file and the repository file are each compared with the BASE revision of the file. The results are displayed in the same view. The differences obtained from the local file comparison are considered outgoing changes and the ones obtained from the repository file comparison are considered incoming changes. If any of the incoming changes overlap outgoing changes then they are in conflict.

A special case of difference is a *diff pseudo-conflict*. This is the case when the left and the right sections are identical but the BASE revision does not contain the changes in that section. By default this type of changes are ignored. If you want to change this you can go to *SVN Preferences* and change the corresponding option.

The right editor of the internal compare view presents either the BASE revision or a revision from the repository of the file so its content cannot be modified. By default when opening a synchronized file in the **Compare** view, a compare is automatically performed. After modifying and saving the content of the local file presented in the left editor, another compare is performed. You will also see the new refreshed status in the ***Working Copy** view*.

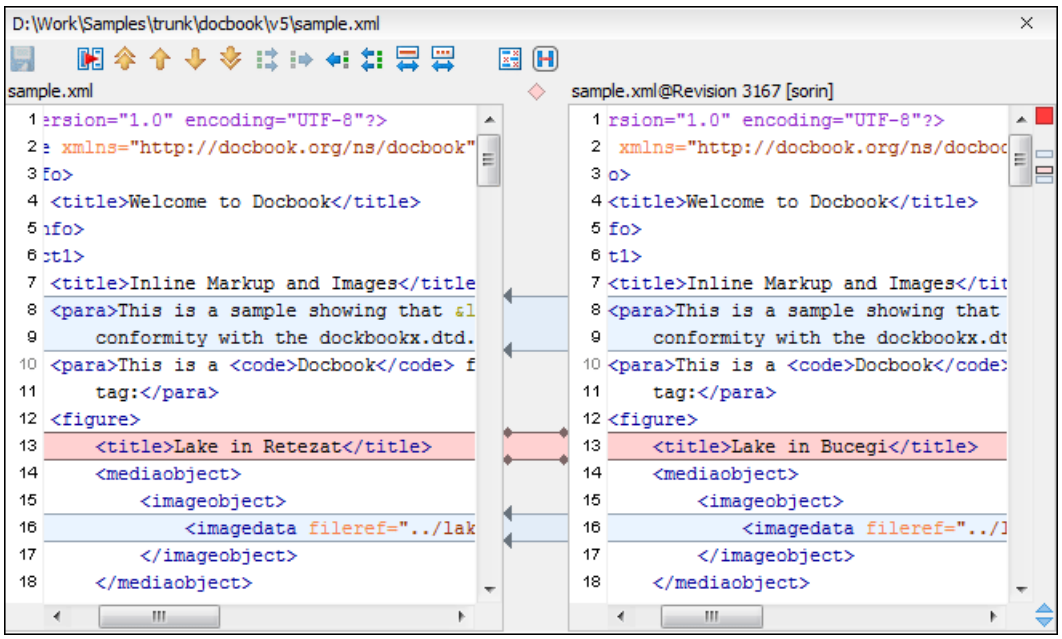


Figure 303: Compare View

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

There are three types of differences:

- incoming changes - Changes committed by other users and not present yet in your working copy file. They are marked with a blue highlight and on the middle divider the arrows point from right to left.
- outgoing changes - Changes you have done in the content of the working copy file. They are marked with a gray highlight and the arrows on the divider are pointing from left to right.
- conflicting changes - This is the case when the same section of text which you already modified in the local file has been modified and committed by some other person. They are marked with a red highlight and red diamonds on the divider.

There are numerous actions and options available in the ***Compare View** toolbar* or in the **Compare** menu from the main menu. You can decide that some changes need adjusting or that new ones must be made. After you perform the adjustments, you may want to perform a new compare between the files. For this case there is an action called **Perform files differencing**. After each files differencing operation the first found change will be selected. You can navigate from one change to another by using the actions **Go to first**, **Go to previous**, **Go to next** and **Go to last modification**. If you decide that some incoming change needs to be present in your working file you can use the action **Copy change from right to left**. This is useful also when you want to override the outgoing modifications contained in a conflicting section. The action **Copy all non-conflicting changes from right to left** copies all incoming changes which are not contained inside a conflicting section in your local file.

Let us assume that only a few words or letters are changed. Considering that the differences are performed taking into account whole lines of text, the change will contain all the lines involved. For finding exactly what words or letters have changed there are available two dialogs which present a more detailed compare result when you double click on the middle divider of a difference: **Word Details** and **Character Details**.

When you want to examine only the changes in the real text content of the files disregarding the changes in the number of white spaces between words or lines there is available an option in the *SVN Preferences* which allows you to enable or disable the white space ignoring feature of the compare algorithm.

Conflicts

A file conflict occurs when two or more developers have changed the same few lines of a file or the properties of the same file. As Subversion knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and try to analyse and resolve the conflicting situation.

Real Conflicts vs Mergeable Conflicts

There are two types of conflicts:

- *real conflict* (decorator in *Name* column) - Syncro SVN Client considers the following resource states to be real conflicts:
  - *conflicted* state - a file reported by SVN as being in this state is obtained after it was updated/merged while having incoming and outgoing content or property changes at the same time, changes which could not be merged. A content conflict (symbol in *Local file status* column) is reported when the modified file has binary content or it is a text file and both local and remote changes were found on the same line. A properties conflict (symbol in *Local properties status* column) is reported when a property's value was modified both locally and remotely;
  - *tree conflicted* state (symbol in *Local file status* column) - obtained after an update or merge operation, while having changes at the directory structure level (for example, file is locally modified and remotely deleted or locally scheduled for deletion and remotely modified);
  - *obstructed* state (symbol in *Local file status* column) - obtained after a resource was versioned as one kind of object (file, directory, symbolic link), but has been replaced outside Syncro SVN Client by a different kind of object.
- *pseudo-conflict* (decorator in *Name* column) - a file is considered to be in *pseudo-conflict* when it contains both incoming and outgoing changes. When incoming and outgoing changes do not intersect, an update operation may automatically merge the incoming file content into the existing locally one. In this case, the *pseudo-conflict* marker is removed. This marker is used only as a warning which should prevent you to run into a real conflict.

Note:

- A conflicting resource cannot be committed to repository. You have to resolve it first, by using **Mark Resolved** action (after manually editing/merging file contents) or by using **Mark as Merged** action (for pseudo-conflicts).
- and decorators are presented only when one of the following view modes is selected: **Modified**, **Incoming**, **Outgoing**, **Conflicts**.
- The marker is used also for folders to signal that they contain a file in real conflict or pseudo-conflict state.

Content Conflicts vs Property Conflicts

A *Content conflict* appears in the content of a file. A merge occurs for every inbound change to a file which is also modified in the working copy. In some cases, if the local change and the incoming change intersect each other, Subversion cannot merge these changes without intervention. So if the conflict is real when updating the file in question the conflicting area is marked like this:

```
<<<<<<< filename
your changes
=======
code merged from repository
>>>>>>> revision
```

Also, for every conflicted file Subversion places three additional temporary files in your directory:

- `filename.ext.mine` - This is your file as it existed in your working copy before you updated your working copy, that is without conflict markers. This file has your latest changes in it and nothing else.
- `filename.ext.rOLDREV` - This is the file that was the BASE revision before you updated your working copy, that is the file revision that you updated before you made your latest edits.

- `filename.ext.rNEWREV` - This is the file that Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD revision of the repository.

OLDREV and NEWREV are revision numbers. If you have conflicts with binary files, Subversion does not attempt to merge the files by itself. The local file remains unchanged (exactly as you last changed it) and you will get `filename.ext.r*` files also.

A *Property conflict* is obtained when two people modify the same property of the same file or folder. When updating such a resource a file named `filename.ext.prej` is created in your working copy containing the nature of the conflict. Your local file property that is in conflict will not be changed. After resolving the conflict you should use the **Mark resolved** action in order to be able to commit the file. Note that the **Mark resolved** action does not really resolve the conflict. It just removes the conflicted flag of the file and deletes the temporary files.

Edit Real Content Conflicts

The conflicts of a file in the conflicted state (a file with the red double arrow icon) can be edited visually with the **Compare** view (the built-in file diff tool) or with an *external diff application*. Resolving the conflict means deciding for each conflict if the local version of the change will remain or the remote one instead of the special conflict markers inserted in the file by the SVN server.

The **Compare** view (or the external diff application *set in Preferences*) is opened with the action **Edit Conflict** which is available on the contextual menus of *the **Working Copy** view* and is enabled only for files in the conflicted state (an update operation was executed but the differences could not be merged without conflicts). The external diff application is called with 3 parameters because it is a 3-way diff operation between the local version of the file from the working copy and the HEAD version from the SVN repository with the BASE version from the working copy as common ancestor.

If *the option **Show warning dialog when edit conflicts** is enabled* you will be warned at the beginning of the operation that the operation will overwrite the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<<<, =======, >>>>>>>) with the original local version of the file that preceded the update operation. If you press the OK button the visual conflict editing will proceed and a backup file of the conflict version received from the SVN server is created in the same working copy folder as the file with the edited conflicts. The name of the backup file is obtained by appending the extension `.sync.bak` to the file as stored on the SVN server. If you press the **Cancel** button the visual editing will be aborted.

The usual operations on the differences between two versions of a file are available on the toolbar of this view:

- **Save** - Saves the modifications of the local version of the file displayed in the left side of the view.
- **Perform Files Differencing** - Applies the diff operation on the two versions of the file displayed in the view. It is useful after modifying the local version displayed in the left side of the view.
- **Go to First Modification** - Scrolls the view to the topmost difference.
- **Go to Previous Modification** - Scrolls the view to the previous difference. The current difference is painted with a darker color than the other ones.
- **Go to Next Modification** - Scrolls the view to the next difference. The current difference is painted with a darker color than the other ones.
- **Go to Last Modification** - Scrolls the view to the last difference.
- **Copy All Non Conflicting Changes from Left to Right** - Not applicable for editing conflicts so it is disabled.
- **Copy Change from Left to Right** - Not applicable for editing conflicts so it is disabled.
- **Copy Change from Right to Left** - Copies the current difference from the left side to the right side by replacing the highlighted text of the current difference from the left side with the one from the right side.
- **Copy All Non Conflicting Changes from Right to Left** - Applies the previous operation for all the differences.
- **Show Modification Details at Word Level** - Displays a more detailed version of the current difference computed at word level.
- **Show Modification Details at Char Level** - Displays a more detailed version of the current difference computed at character level.
- **Ignore Whitespaces** - The text nodes are normalized before computing the difference so that if two text nodes differ only in whitespace characters they are reported as equal.

The operation begins by overwriting the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<<<, =======, >>>>>>>) with the original local version of the file before running the update action which created the conflict. After that the differences between this original local version and the repository version are displayed in the **Compare** view.

If you want to edit the conflict version of the file directly in a text editor instead of the visual editing offered by the **Compare** view you should work on the local working copy file after the update operation without running the action **Edit Conflict**. If you decide that you want to edit the conflict version directly after running the action **Edit Conflict** you have to work on the `.sync.bak` file.

If you did not finish editing the conflicts in a file at the first run of the action **Edit Conflict** you can run the action again and you will be prompted to choose between resuming the editing where the previous run left it and starting again from the conflict file received from the SVN server.

After the conflicts are edited and saved in the local version of the file you should run:

- either the action **Mark Resolved** on the file so that the result of the conflict editing process can be committed to the SVN repository,
- or the action **Revert** so that the repository version overwrites all the local modifications.

Both actions remove the backup file and other temporary files created with the conflict version of the local file.

Revert Your Changes

If you want to undo all changes you made in a file since the last update you need to select the file, right click to pop up the contextual menu and then select **Revert**. A dialog will pop up showing you the files that you have changed and can be reverted. Select those you want to revert and click the **OK** button. Revert will only undo your local changes. It does not undo any changes which have already been committed. If you choose to revert the file to the pristine copy which resides in the administration folders then the eventual conflict is solved by losing your outgoing modifications. If you try to revert a resource not under version control, the resource will be deleted from the file system.

If you want some of your outgoing changes to be overridden you must first open the file in ***Compare** view* and choose the sections to be replaced with ones from the repository file. This can be achieved either by editing directly the file or by using the action **Copy change from right to left** from the ***Compare** view toolbar*. After editing the conflicting file you have to run the action **Mark as merged** before committing it.

If you want to drop all local changes and in the same time bring all incoming changes into your working copy resource you can use the **Override and update** action which discards the changes in the local file and updates it from the repository. A dialog will show you the files that will be affected.

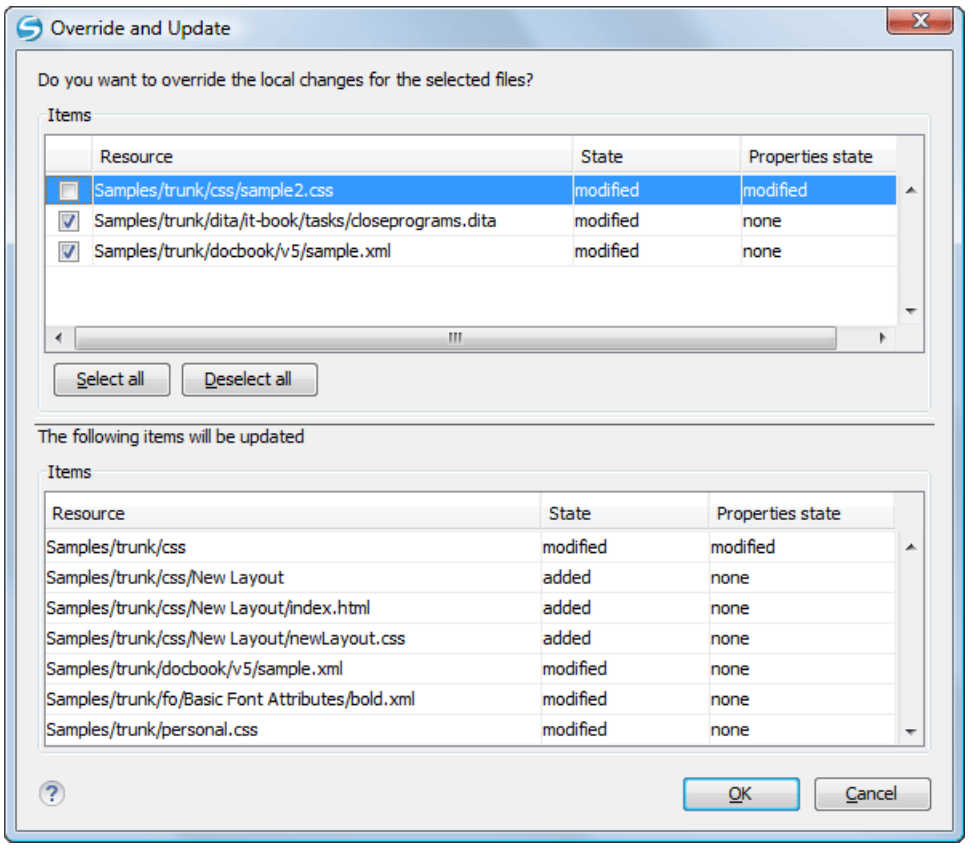


Figure 304: Override and update dialog

In the first table in the dialog you will be able to see the resources that will be overridden. You can also select or deselect them as you wish. In the second table you will find the list of resources that will be updated. Only resources that have an incoming status are updated.

Merge Conflicted Resources

Before you can safely commit your changes to the repository you must first resolve all conflicts. In the case of pseudo-conflicts they can be resolved in most cases with an update operation which will merge the incoming modifications into your working copy resource. In the case of real conflicts, conflicts that persist after an update operation, it is necessary to resolve the conflict using the built-in compare view and editor or, in the case of properties conflict, the ***Properties*** *view*. Before you can commit you must *mark as resolved* the affected files.

Both pseudo and real conflicts can be resolved without an update. You should open the file in the compare editor and decide which incoming changes need to be copied locally and which outgoing changes must be overridden or modified. After saving your local file you have to use the *Mark as merged* action from the contextual menu before committing.

Drop Incoming Modifications

In the situation when your file is in conflict but you decide that your working copy file and its content is the correct one, you can decide to drop some or all of the incoming changes and commit afterwards. The action **Mark as merged** proves to be useful in this case too. After opening the conflicting files with ***Compare*** *view*, *Editor* or editing their properties in the **Properties** view and deciding that your file can be committed in the repository replacing the existing one, you should use the **Mark as merged** action. When you want to override completely the remote file with the local file you should run the action **Override and commit** which drops any remote changes and commits your file.

In general it is much safer to analyze all incoming and outgoing changes using the **Compare** view and only after to update and commit.

Tree Conflicts

A *tree conflict* is a conflict at the directory tree structure level and occurs when the user runs an update action on a resource that:

- it is locally modified and the same resource was deleted from the repository (or deleted as a result of being renamed or moved);
- it was locally deleted (or deleted as a result of being renamed or moved) and the same resource is incoming as modified from the repository.

The same conflict situation can occur after a merge or a switch action. The action ends with an error and the folder containing the file that is now in the tree conflict state is also marked with a conflict icon.

Such a conflict can be resolved in one of the following ways which are available when the user double clicks on the conflicting resource or when running the **Edit conflict** action:

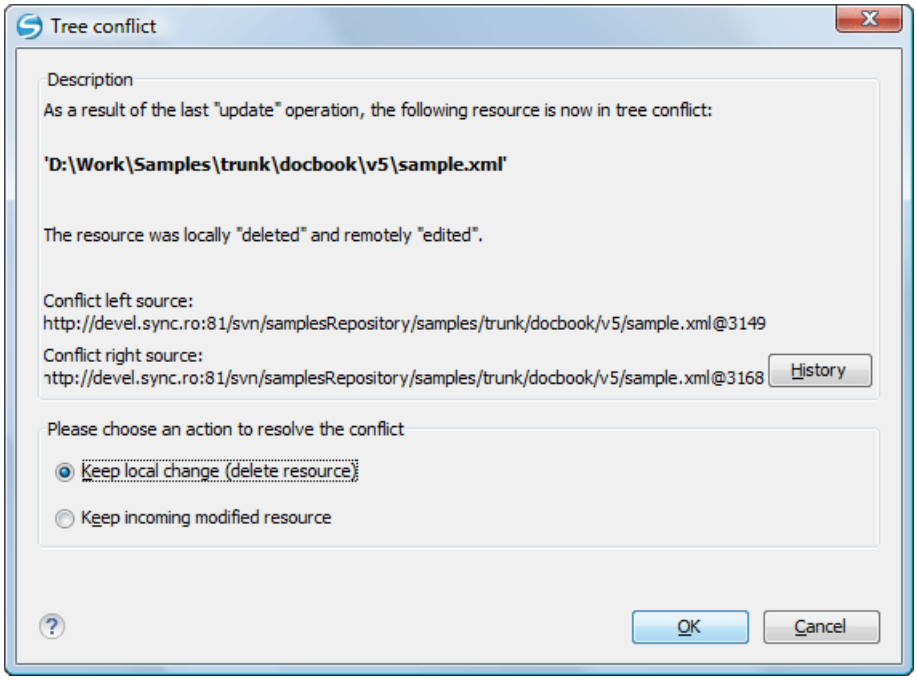


Figure 305: Resolve a tree conflict

- Keep the local modified file - If there is a renamed version of the file committed by other user that will be added to the working copy too.
- Delete the local modified file - Keeps the incoming change that comes from the repository.

Update the Working Copy

While you are working on a project, other members of your team may be committing changes to the project repository. To get these changes, you have to *update* your working copy. Updating may be done on single files, a set of selected files, or recursively on entire directory hierarchies. The update operation can be performed from ***Working Copy** view*. It updates the selected resources to the last synchronized revision (if remote information is available) or to the *HEAD* revision of the repository.

There are three different kinds of incoming changes:

- *Non-conflicting* - A non-conflicting change occurs when a file has been changed remotely but has not been modified locally.
- *Conflicting, but auto-mergeable* - An auto-mergeable conflicting change occurs when a text file has been changed both remotely and locally (i.e. has non-committed local changes) but the changes are on different lines of text. Not applicable to binary resources (for example multimedia files, PDFs, executable program files)
- *Conflicting* - A conflicting change occurs when one or more of the same lines of a text file have been changed both remotely and locally.

If the resource contains only incoming changes or the outgoing changes do not intersect with incoming ones then the update will end normally and the Subversion system will merge incoming changes into the local file. In the case of a conflicting situation the update will have as result a file with conflict status.

The Syncro SVN Client allows you to update your working copy files to a specific revision, not only the most recent one. This can be done by using the **Update to revision/depth** action from the **Working Copy** view (**All Files** view mode) or the **Update to revision** action from the ***History*** *view* contextual menu.

If you select multiple files and folders and then you perform an **Update** operation, all of those files and folders are updated one by one. The Subversion client makes sure that all files and folders belonging to the same repository are updated to the exact same revision, even if between those updates another commit occurred.

When the update fails with a message saying that there is already a local file with the same name Subversion tried to checkout a newly versioned file, and found that an unversioned file with the same name already exists in your working folder. Subversion will never overwrite an unversioned file unless you specifically do this with an **Override and update** action. If you get this error message, the solution is simply to rename the local unversioned file. After completing the update, you can check whether the renamed file is still needed.

Send Your Changes to the Repository

Sending the changes you made to your working copy is known as *committing* the changes. If your working copy is up to date and there are no conflicts, you are ready to commit your changes.

The **Commit** action sends the changes in your local working copy to the repository. After selecting the action from the contextual menu you will see a dialog displaying the resources that can be committed.

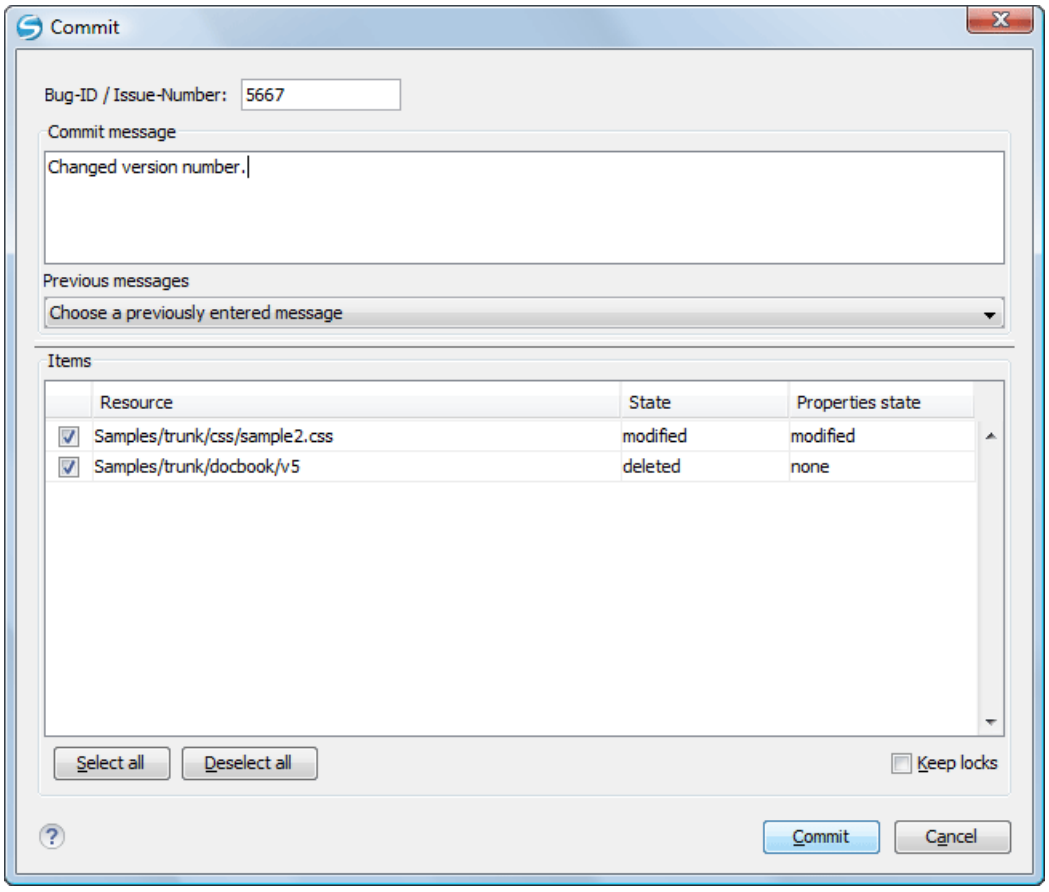


Figure 306: Commit dialog

Enter a comment to associate with the commit or choose a previously entered comment from the list (the last 10 commit messages will be remembered even after restarting the SVN client application). The dialog will list modified, added, deleted and unversioned resources. All modified, added and deleted resources will be selected by default. If you don't

want a changed file to be committed, just uncheck that file. The unversioned items are not selected by default unless you have selected them specifically before issuing the commit command.

To select all resources, click **Select All**. To deselect all resources, click **Deselect All**. Checking the **Keep locks** option will preserve any locks you have on repository resources. Your working copy must be up-to-date with respect to the resources you are committing. This is ensured by using the **Update** action prior to committing, resolving conflicts and re-testing as needed. If your working copy resources you are trying to commit are *out of date* you will get an appropriate error message.

The table presented in the dialog is sortable. For example if you want to see all the resources that are in the *modified* state click on the **State** column header to sort the table by that column.

The modifications that will be committed for each file can be reviewed in the compare editor window by double clicking on the file in the **Commit** dialog or by right clicking and selecting the action **Show Modifications** from the contextual menu.

If you have modified files which have been included from a different repository using `svn:externals`, those changes cannot be included in the same commit operation.

Integration with Bug Tracking Tools

Users of bug tracking systems can associate the changes they make in the repository resources with a specific ID in their bug tracking system. The only requirement is that the user includes the bug ID in the commit message that he enters in the **Commit** dialog. The format and the location of the ID in the commit message are configured with SVN properties.

To make the integration possible Syncro SVN Client needs some data about the bug tracking tool used in the project. You can configure this using the following *SVN properties* which must be set on the folder containing resources associated with the bug tracking system. Usually they are set recursively on the root folder of the working copy.

- **bugtraq:message** - A string property. If it is set *the **Commit** dialog* will display a text field for entering the bug ID. It must contain the string *%BUGID%*, which is replaced with the bug number on commit.
- **bugtraq:label** - A string property that sets the label for the text field configured with the **bugtraq:message** property.
- **bugtraq:url** - A string property that is the URL pointing to the bug tracking tool. The URL string should contain the substring *%BUGID%* which Syncro SVN Client replaces with the issue number. That way the resulting URL will point directly to the correct issue.
- **bugtraq:warnifnoissue** - A boolean property with the values *true*/*yes* or *false*/*no*. If set to *true*, the Syncro SVN Client will warn you if the bug ID text field is left empty. The warning will not block the commit, only give you a chance to enter an issue number.
- **bugtraq:number** - A boolean property with the value *true* or *false*. If this property is set to *false*, then any character can be entered in the bug ID text field. If the property is set to *true* or is missing then only numbers are allowed as the bug ID.
- **bugtraq:append** - A boolean property. If set to *false*, then the bug ID is inserted at the beginning of the commit message. If *yes* or not set, then it's appended to the commit message.
- **bugtraq:logregex** - This property contains one or two regular expressions, separated by a newline. If only one expression is set, then the bug ID's must be matched in the groups of the regular expression string, for example `[Ii]ssue #?(\d+)` If two expressions are set, then the first expression is used to find a string which relates to a bug ID but may contain more than just the bug ID (e.g. `Issue #123` or `resolves issue 123`). The second expression is then used to extract the bug ID from the string extracted with the first expression. An example: if you want to catch every pattern `issue #XXX` and `issue #890, #789` inside a log message you could use the following strings:
  - `[Ii]ssue #?(\d+)(,? ?#?(\d+))+`
  - `(\d+)`

The data configured with these SVN properties is stored on the repository when a revision is committed. A bug tracking system or a statistics tools can retrieve from the SVN server the revisions that affected a bug and present the commits related to that bug to the user of the bug tracking system.

If the **bugtraq:url** property was filled in with the URL of the bug tracking system and this URL includes the *%BUGID%* substring as specified above in the description of the **bugtraq:url** property then *the **History** view* presents the bug ID as a hyperlink in the commit message. A click on such a hyperlink in the commit message of a revision opens a Web browser at the page corresponding to the bug affected by that commit.

Obtain Information for a Resource

This section explains how to obtain information for a SVN resource:

Request Status Information for a Resource

While you are working you often need to know which files you have changed, added, removed or renamed, or even which files got changed and committed by others. That's where the **Synchronize** action from ***Working Copy** view* comes in handy. The **Working Copy** view will show you every file that has changed in any way in your working copy, as well as any unversioned files you may have.

If you want more detailed information about a given resource you can use the **Information** action from the **Working Copy** view *contextual menu*. A dialog called **SVN Information** will pop up showing remote and local information regarding the resource, such as:

- local path and repository location
- revision number
- last change author, revision and date
- commit comment
- information about locks
- local file status
- local properties status
- remote file status
- remote properties status
- file size, etc.

The value of a property of the resource displayed in the dialog can be copied by right clicking on the property and selecting the **Copy** action.

A less detailed list of information is also presented when you hover with the mouse pointer over a resource and the tooltip window is displayed.

Request History for a Resource

In Subversion, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you should load the history data of a resource in *the **History** view* by running the **Show History** action. This action is available in the **Working Copy** and **Repositories** views and also from the menus with the same name.

From the **Repositories** view you can display the log history regarding any remote resource residing in repository. From the **Working Copy** view you can display the history of local versioned resources and newly incoming resources.

History View

In Subversion, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from any of the following three views: *Repository view menu* or *Working copy view menu*. From the **Working copy view** you can display the history of local versioned resources.

The view consists of three distinct areas:

- The revision table showing revision numbers, date/time of revision, the author's name, as well as the first line of the commit message;
- The list of resources affected by this revision (modified, added, deleted or changed properties);

- The commit message for the selected revision.

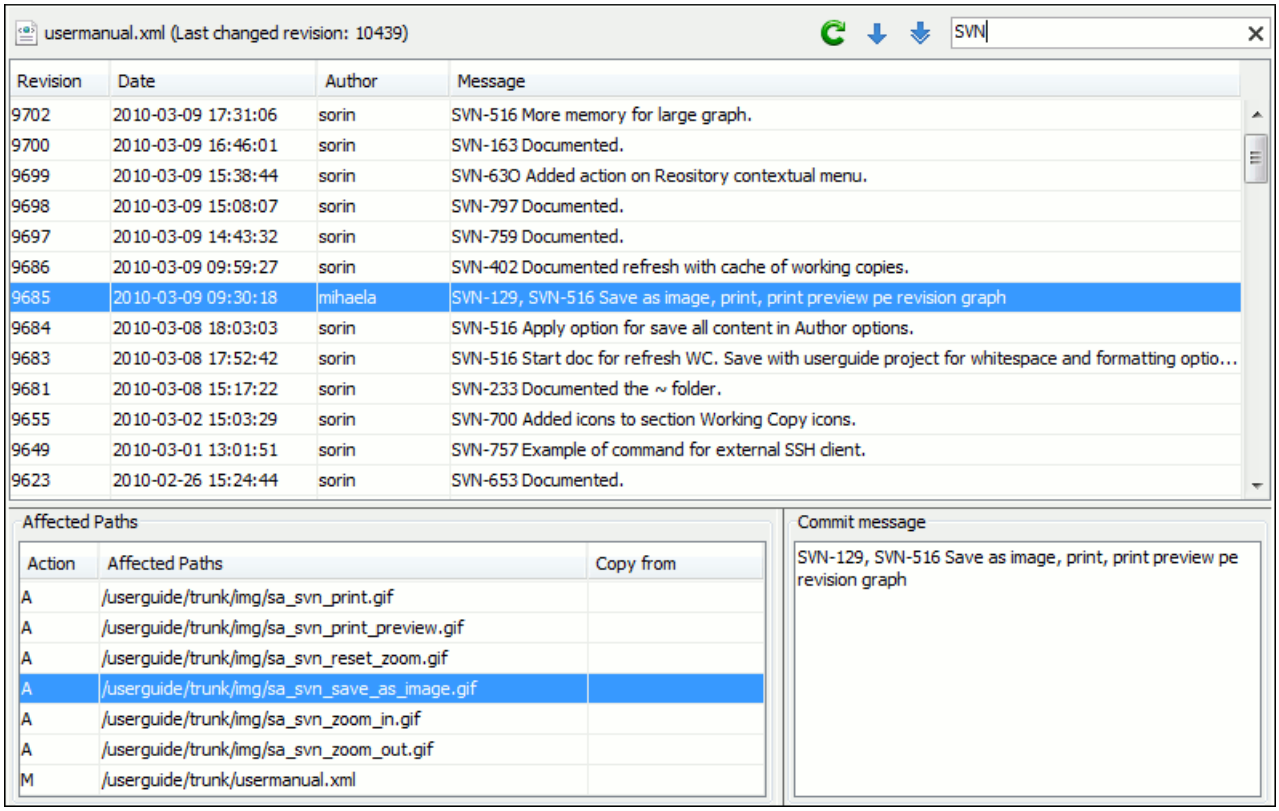


Figure 307: History View

The Contextual Menu of the History View

When a single resource is selected in the **History** view, the contextual menu contains the following actions:

- **Compare** - Compares the selected revision with your working copy file. It is enabled only when you select a file.
- **Open** - Opens the selected revision of the file into the Editor. This action is enabled only if the resource is a file.
- **Open With...** - Displays the ***Open with...*** dialog for specifying the editor in which the selected file revision is opened. This action is enabled only if the resource is a file.
- **Get Contents** - Replaces the current content of the local version of the selected file with the content of the selected revision.
- **Save revision to...** - Saves the selected revision to a file so you have an older version of that file. This option is available only when you access the history of a file, and it saves a version of that one file only.
- **Revert changes from this revision** - Reverts changes which were made in the selected revision. The changes are reverted in your working copy so this operation does not affect the repository file. The action undoes the changes made only in selected revision. It does not replace your working copy file with the entire file at the earlier revision. This action is useful for undoing an earlier change when other unrelated changes have been made since the date of the revision. This action is enabled when the resource history was launched for a local working copy resource.
- **Update to revision** - Updates your working copy resource to the selected revision. Useful if you want to have your working copy reflect a time in the past. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy is inconsistent and you are unable to commit your changes.
- **Check out from revision...** - Gets the content of the selected revision into local file system.
- **Show Annotation...** - Brings up a dialog for selecting the start revision and the end revision of the interval of revisions for which the SVN annotations is computed and marked in the selected resource in the editor panel. This action is available only when you access the history of a file.
- **Change** - Allows you to change commit data for a file:
  - *Author* - Allows you to change the user name that committed the selected file revision.

- *Message* - Allows you to change the commit message of the selected file revision.

When two resources are selected in the **History** view, the contextual menu contains the following actions:

- **Compare revisions** - When the resource is a file the action compares the two selected revisions using the **Compare** view. When the resource is a folder, the action *displays the set of all resources from that folder that were changed* between the two revision numbers.
- **Revert changes from these revisions** - Similar to the `svn-merge` command, it merges two selected revisions into the working copy resource. This action is only enabled when the resource history was requested for a working copy item.

Directory Change Set View

The result of comparing two reference revisions from the history of a folder resource is a set with all the resources changed between the two revision numbers. The changed resources can be contained in the folder or in a subfolder of that folder. These resources are presented in a tree format. For each changed resource all the revisions committed between the two reference revision numbers are presented.

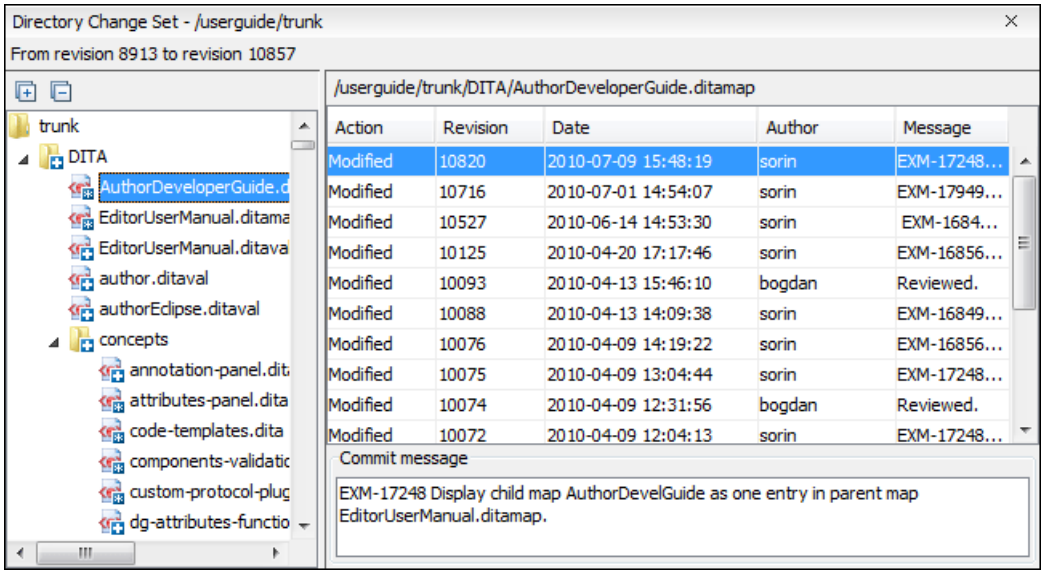


Figure 308: Directory Change Set View

The set of changed resources displayed in the tree is obtained by running the action **Compare revisions** available on the context menu of the **History** view when two revisions of a folder resource are selected in the **History** view.

The left side panel of the view contains the tree hierarchy with the names of all the changed resources between the two reference revision numbers. The right side panel presents the list with all the revisions of the resource selected in the left side tree. These revisions were committed between the two reference revision numbers. Selecting one revision in the list displays the commit message of that revision in the bottom area of the right side panel.

A double click on a file listed in the left side tree performs a diff operation between the two revisions of the file corresponding to the two reference revisions. A double click on one of the revisions displayed in the right side list of the view performs a diff operation between that revision and the previous one of the same file.

The context menu of the right side list contains the following actions:

- **Compare with previous version** - Performs a diff operation between the selected revision in the list and the previous one.
- **Open** - Opens the selected revision in the associated editor type.
- **Open with...** - *Displays a dialog* with the available editor types and allows the user to select the editor type for opening the selected revision.
- **Save revision to...** - Saves the selected revision in a file on disk.
- **Show Annotation** - Requests the annotations of the file and *displays them in the **Annotations** view*.

Management of SVN Properties

In the *Properties view* you can read and set the Subversion properties of a file or folder. There is a set of predefined properties with special meaning to Subversion. For more information about properties in Subversion see the SVN Subversion specification. Subversion properties are revision dependent. After you change, add or delete a property for a resource, you have to commit your changes to the repository.

If you want to change the properties of a given resource you need to select that resource from the ***Working Copy** view* and run the **Show properties** action from the contextual menu. The **Properties** view will show the local properties for the resource in the working copy. Once the Properties view is visible, it will always present the properties of the currently selected resource. In the **Properties** view *toolbar* there are available actions which allow you to add, change and delete the properties.

If you choose the **Add a new property** action, a new dialog will pop-up containing:

- **Name** - Combo box which allows you to enter the name of the property. The drop down list of the combo box presents the predefined Subversion properties such as **svn:ignore**, **svn:externals**, **svn:needs-lock**, etc.
- **Current value** - Text area which allows you to enter the value of the new property.

If the selected item is a directory, you can also set the property recursively on its children by checking the **Set property recursively** checkbox.

If you want to change the value for a previously set property you can use the **Edit property** action which will display a dialog where you can set:

- **Name** - Property name (cannot be changed).
- **Current value** - Presents the current value and allows you to change it.
- **Base value** - The value of the property, if any, from the resource in the pristine copy. It cannot be modified.

If you want to completely remove a property previously set you can choose the **Remove property** action. It will display a confirmation dialog in which you can choose also if the property will be removed recursively.

In the ***Properties** view* there is a **Refresh** action which can be used when the properties have been changed from outside the view. This can happen, for example, when the view was already presenting the properties of a resource and they have been changed after an **Update** operation.

Branches and Tags

One of the fundamental features of version control systems is the ability to create a new line of development from the main one. This new line of development will always share a common history with the main line if you look far enough back in time. This line is known as a branch. Branches are mostly used to try out features or fixes. When the feature or fix is finished, the branch can be merged back into the main branch (trunk).

Another feature of version control systems is the ability to take a snapshot of a particular revision, so you can at any time recreate a certain build or environment. This is known as tagging. Tagging is especially useful when making release versions.

In Subversion there is no difference between a tag and a branch. On the repository both are ordinary directories that are created by copying. The trick is that they are cheap copies instead of physical copies. Cheap copies are similar to hard links in Unix, which means that they merely link to a specific tree and revision without making a physical copy. As a result branches and tags occupy little space on the repository and are created very quickly.

As long as nobody ever commits to the directory in question, it remains a tag. If people start committing to it, it becomes a branch.

Create a Branch / Tag

In the ***Working Copy** view* or in the ***Repositories** view*, select the resource which you want to copy to a branch or tag, then select the command *Branch / Tag...* from the **Tools** menu.

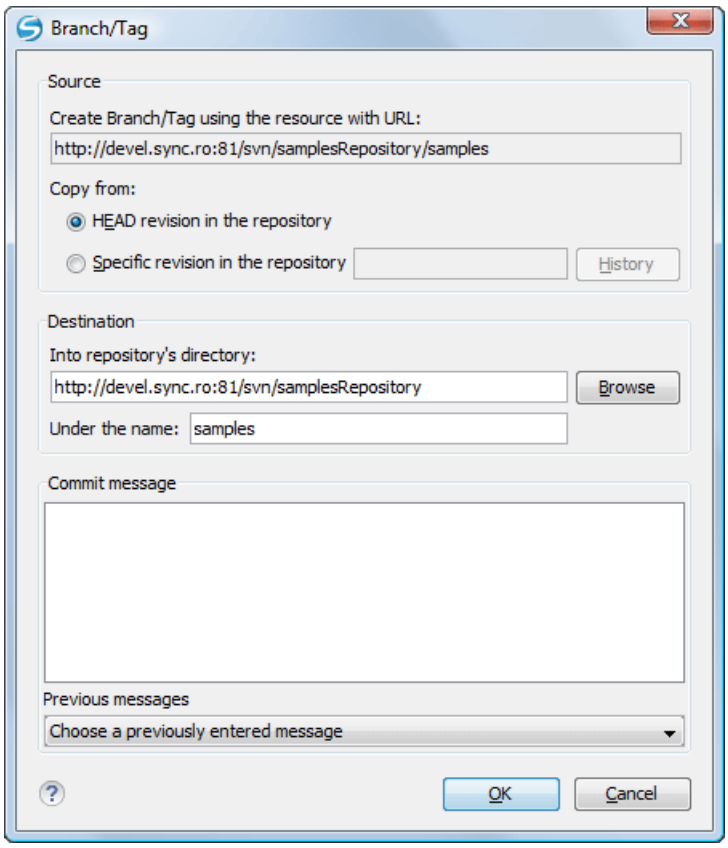


Figure 309: The Branch / Tag dialog

The default target URL for the new branch / tag will be the repository URL of the selected resource from your working copy, divided in two: the URL of the parent and the selected resource's name. You may specify other resource name if you want to make a branch / tag using a different name than the one of the selected resource, by modifying the field labeled **Under the name:**. The new branch / tag will be created as child of the specified repository directory URL and having the new provided name. To change the parent directory URL to the new path for your branch / tag, click on the **Browse** button and choose a repository target directory for your resource.

You can also specify the source of the copy. There are three options:

- **HEAD revision in the repository** - The new branch / tag will be copied in the repository from the HEAD revision. The branch will be created very quickly as the repository will make a cheap copy.
- **Specific revision in the repository** - The new branch will be copied in the repository but you can specify exactly the desired revision. This is useful for example if you forgot to make a branch / tag when you released your application. If you click on the **History** button on the right you can select the revision number from *the **History** dialog*. This type of branch will also be created very quickly.
- **Working copy** - The new branch will be a copy of your local working copy. If you have updated some files to an older revision in your working copy, or if you have made local changes, that is exactly what goes into the copy. This involves transferring some data from your working copy back to the repository, more exactly the locally modified files.

When you are ready to create the new branch / tag, write a commit comment in the corresponding field and press the **OK** button.

Merging

At some stage during the development you will want to merge the changes made on one branch back into the trunk, or vice versa. Merge is closely related to Diff. The merge is accomplished by comparing two points (branches or revisions) in the repository and applying the obtained differences to your working copy.

It is a good idea to perform a merge into an unmodified working copy. If you have made changes to your working copy, commit them first. If the merge does not go as you expect, you may want to revert the changes and revert cannot recover your uncommitted modifications.

The **Merge** action can be found in the **Tools** menu of Syncro SVN Client. The directory selected when you issued the command will be the result directory of the merge operation.

There are three common types of merging which are handled in different ways:

- merge revisions - integrate the modifications from a branch into the trunk or a different branch, when the branches are created from the same trunk;
- reintegrate a branch - integrate the modifications from a branch into the trunk;
- merge two different trees - the general case of integrating some changes between two different branches.

Merge Revisions

This is the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch or trunk. An example of such operation can be the following: calculate the changes necessary to get (from) revision 17 of branch B1 (to) revision 25 of branch B1, and apply those changes to my working copy, of the trunk or another branch.

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:

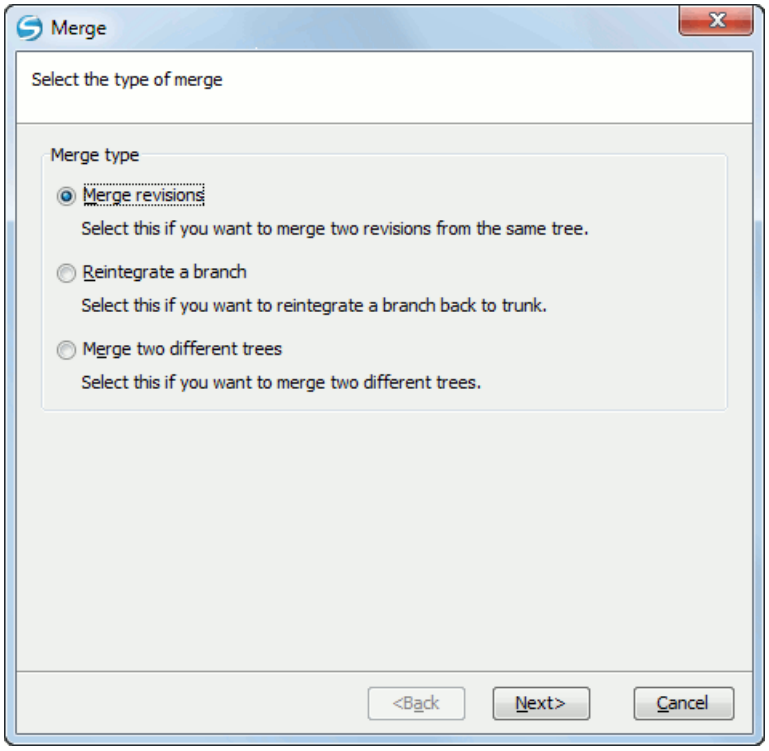


Figure 310: The Merge Wizard - The Merge Type

2. Select the option **Merge revisions**.
3. Press the **Next** button.
   The second step of the **Merge** wizard is displayed:

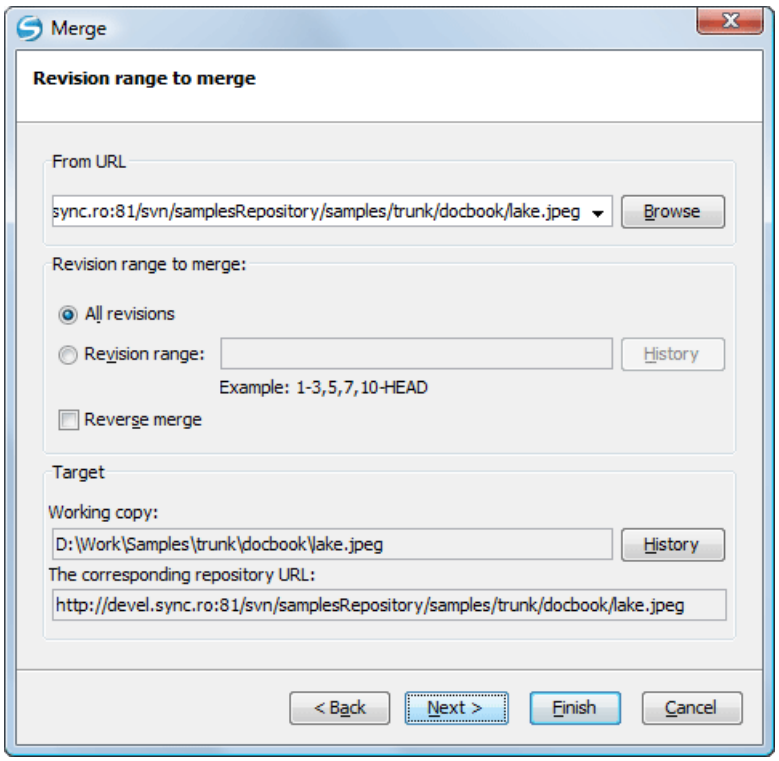


Figure 311: The Merge Wizard - Revisions Range

4. In the **From URL** field enter the folder URL of the branch or tag containing the changes that you want to port into your working copy.

   You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

5. Select the revision range.
   a) Choose between all revisions and a revision range.

      In the **Revision range to merge** section you can choose to merge all revisions or enter the list of revisions you want to merge in the **Revision range** field. This can be a single revision, a list of comma separated specific revisions, or a range of revisions separated by a dash, or any combination of these.

      The **History** button opens the *the **History** dialog* which allows selecting the list of revisions to be merged in the easiest way. One or several revisions can be selected in that dialog.

      Be careful about using the HEAD revision. It may not refer to the revision you think it does if someone else made a commit after your last update.

   b) Check the **Reverse merge** box (optional).

      If you want to merge changes back out of your working copy, to revert a change which has already been committed, select the revisions to revert and check the **Reverse merge** box.

6. Specify the target where the changes of the branch will be integrated.
   a) Enter the working copy folder in the **Working copy** field.

      By default the root folder of the current working copy from the **Working Copy** view is set in this field.

      If you have already merged some changes from this branch and you remember the last merged revision, you can select that revision for the working copy using the **History** button. For example, if you have merged revisions 27 to 33 last time, then the start point for this merge operation should be revision 33.

   b) Specify the URL of the target branch that will receive the changes.

By default the URL of the repository of the current working copy from the **Working Copy** view is set in this field.

Subversion has merge tracking features and automatically records the last merged revision so you do not need to remember when you performed the last merge.

7. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:

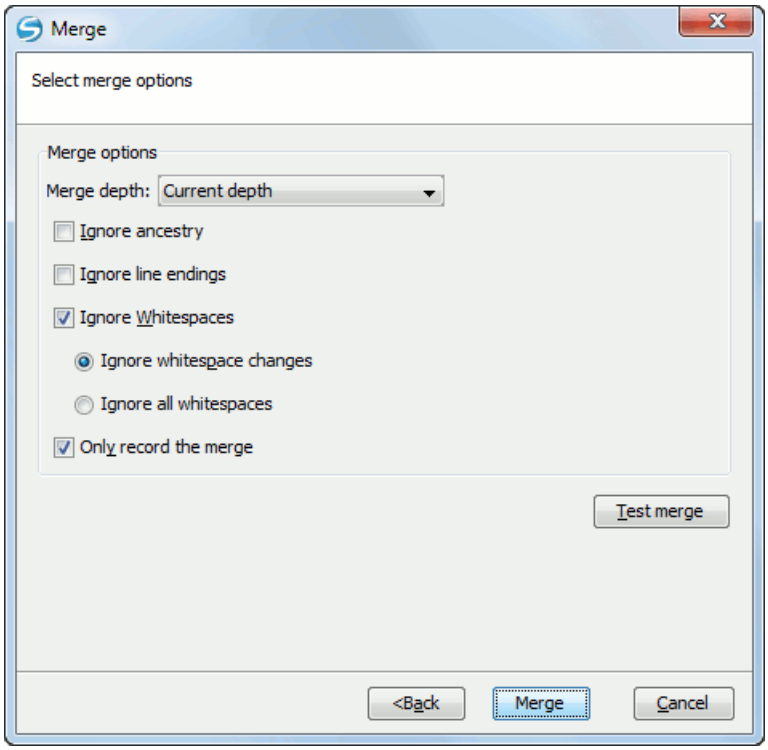


Figure 312: The Merge Wizard - Advanced Options

8. Set advanced options if necessary before starting the merge process.
   a) Set the depth of the merge operation in the **Merge depth** combo box.

      You can specify how far down into your working copy the merge should go by selecting one of the following values:

      - Current depth
      - Recursive (infinity)
      - Immediate children (immediates)
      - File children only (files)
      - This folder only (empty)

      The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

   b) Check the **Ignore ancestry** checkbox (optional).

      The **Ignore ancestry** checkbox allows a merge to be applied between a branch and the trunk or between two branches even if they do not share a common ancestry. Normally the branch and the trunk or the two branches that are merged must have a common ancestor revision in the same repository. In case the two merged trees were imported in the repository they are not related in the sense of a common ancestor tree and the merge operation is possible by ignoring the missing common ancestry of the two merged trees.

   c) Check the **Ignore line endings** checkbox (optional).
   d) Check the **Ignore Whitespaces** checkbox (optional).

The **Ignore line endings** and **Ignore whitespaces** checkboxes allow you to specify how the line endings and whitespace changes should be handled. If they are checked the changes due only to the line endings and whitespaces are ignored. The default behavior is to treat all whitespace and line-end differences as real changes to be merged. **Ignore whitespace changes** excludes changes which are caused by a change in the amount or type of whitespace, for example changing the indentation or changing tabs to spaces. Adding whitespace where there was none before, or removing a whitespace completely is still shown as a change. If **Ignore all whitespaces** is checked all whitespace-only changes are excluded.

e) Check the **Only record the merge** checkbox (optional).

If you are using merge tracking support and you want to mark a revision as having been merged, without actually doing the merge here, check the **Only record the merge** checkbox. You might want to do this for two possible reasons. You make the changes by hand, then mark the change as merged so that the merge tracking algorithm is aware of it. Or you might want to prevent a particular revision from being merged by marking it as already merged. This will prevent future merging.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you do a dry run of the merge operation in order to see what files are affected and how without modifying the working copy at all. This is very helpful in detecting where conflicts may occur.

9. Press the **Merge** button.
The merge operation is executed.

10. Optionally resolve the conflicts that were created by the merge operation.

After the merge operation is finished it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.

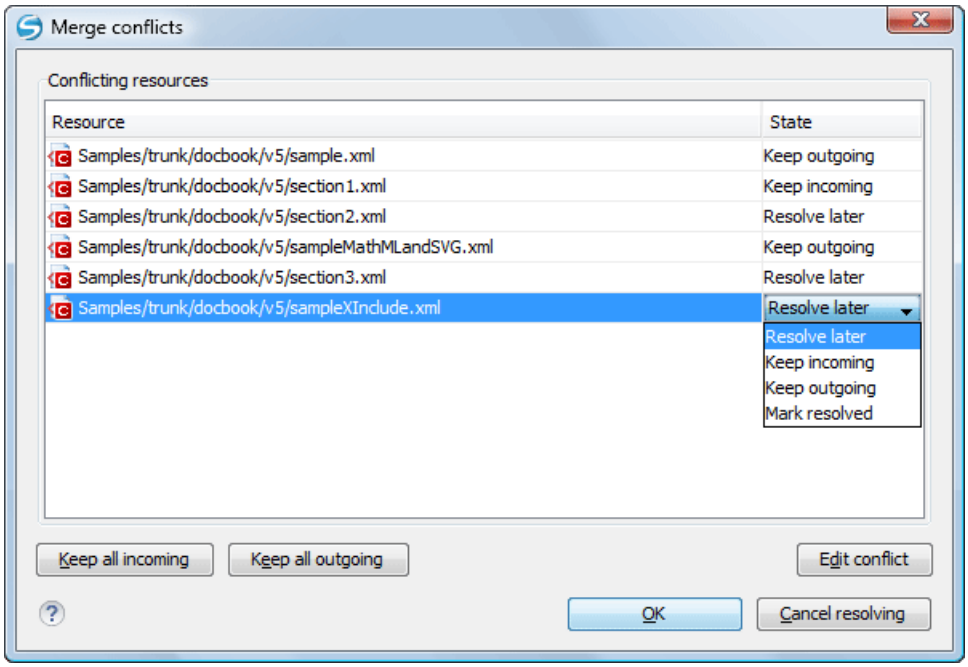


Figure 313: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used to leave the conflict as it is for manual resolving it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.

- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do that, use the **Edit conflict** button, which will bring to you a dialog presenting the conflicting resource's content for current working copy version and the one with the incoming modifications. After manually resolving the conflict, the resource will be marked as resolved.

When the merge is completed it's a good idea to look at the result of the merge in the specified working copy and see if it meets your expectations. Because merging is sometimes complicated, when there are major changes, *conflicts may appear*.

Reintegrate a Branch

There are some conditions which apply to a reintegrate merge: Firstly, the server must support merge tracking. The working copy must be of depth infinite (no sparse checkouts), and it must not have any local modifications, switched items or items that have been updated to revisions other than HEAD. All changes to trunk made during branch development must have been merged across to the branch (or marked as having been merged).

- The server must support merge tracking.
- The working copy must be of depth infinite (no sparse checkouts), and it must not have any local modifications, switched items or items that have been updated to revisions other than HEAD.
- All changes to trunk made during branch development must have been merged across to the branch (or marked as having been merged).

This method covers the case when you have made a feature branch. All trunk changes have been ported to the feature branch, and now you want to merge it back into the trunk. Because you have kept the feature branch synchronized with the trunk, the latest versions of branch and trunk will be absolutely identical except for your branch changes. These changes can be reintegrated into the trunk by this method.

It uses the merge-tracking features of Subversion to calculate the correct revision ranges and to perform additional checks which ensure that the branch has been fully updated with trunk changes. The range of revisions to merge will be calculated automatically. This ensures that you don't accidentally undo work that others have committed to trunk since you last synchronized changes. After the merge, all branch development has been completely merged back into the main development line. The branch is now redundant and can be deleted.

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:

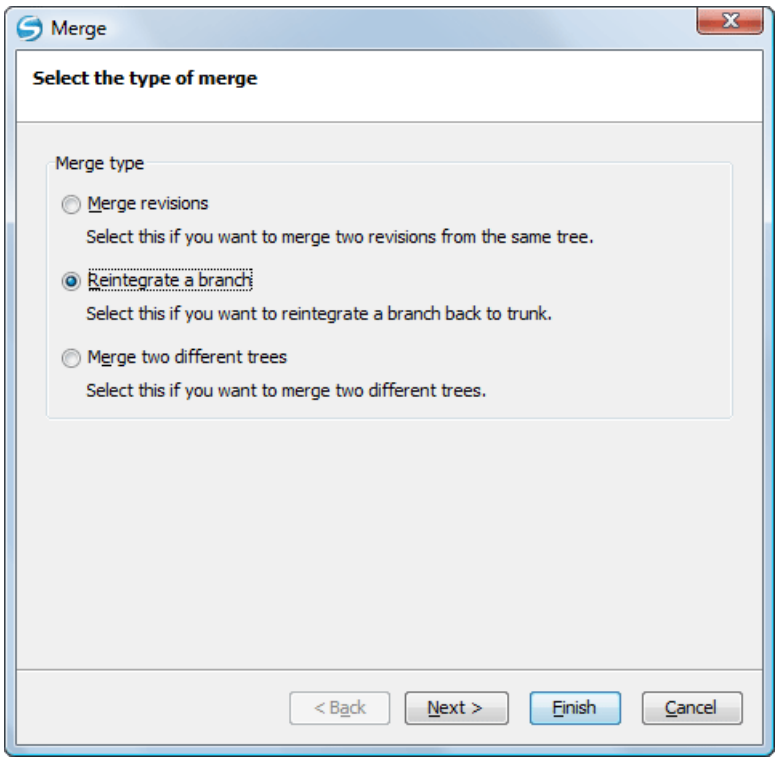


Figure 314: The Merge Wizard - The Merge Type

2. Select the option **Reintegrate a branch**.
3. Press the **Next** button.
   The second step of the **Merge** wizard is displayed:

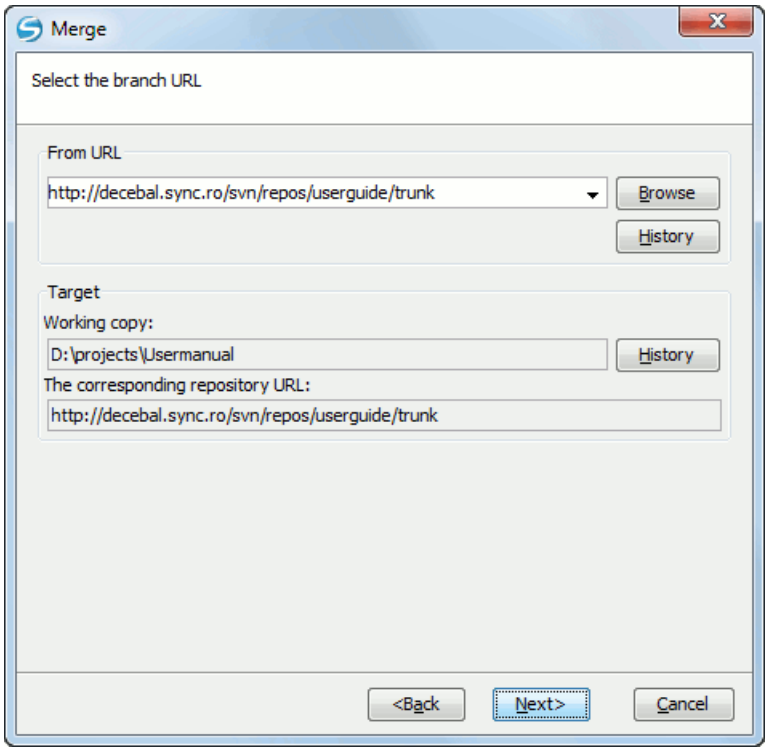


Figure 315: The Merge Wizard - Reintegrate Branch

4. In the **From URL** field enter the folder URL of the branch or tag containing the changes that you want to integrate.

   You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

   The History button opens *the **History** dialog* which allows you to select a revision number of the repository with the changes.

5. Select the target of the operation.
   a) Select the path of the working copy.
   b) Select the URL of the repository corresponding to the working copy.

   The target panel of the dialog reminds you the location of the target resource from the working copy where the merge result will be saved and its corresponding repository URL.

6. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:

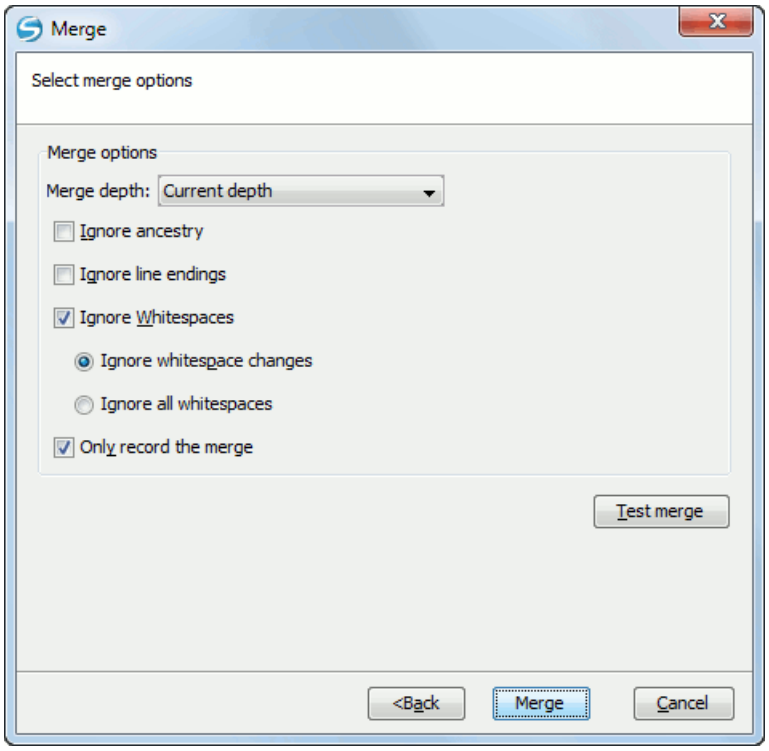


Figure 316: The Merge Wizard - Advanced Options

7. Set advanced options if necessary before starting the merge process.
   a) Set the depth of the merge operation in the **Merge depth** combo box.

      You can specify how far down into your working copy the merge should go by selecting one of the following values:

      - Current depth
      - Recursive (infinity)
      - Immediate children (immediates)
      - File children only (files)
      - This folder only (empty)

      The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

   b) Check the **Ignore ancestry** checkbox (optional).

The **Ignore ancestry** checkbox allows a merge to be applied between a branch and the trunk or between two branches even if they do not share a common ancestry. Normally the branch and the trunk or the two branches that are merged must have a common ancestor revision in the same repository. In case the two merged trees were imported in the repository they are not related in the sense of a common ancestor tree and the merge operation is possible by ignoring the missing common ancestry of the two merged trees.

c) Check the **Ignore line endings** checkbox (optional).

d) Check the **Ignore Whitespaces** checkbox (optional).

The **Ignore line endings** and **Ignore whitespaces** checkboxes allow you to specify how the line endings and whitespace changes should be handled. If they are checked the changes due only to the line endings and whitespaces are ignored. The default behavior is to treat all whitespace and line-end differences as real changes to be merged. **Ignore whitespace changes** excludes changes which are caused by a change in the amount or type of whitespace, for example changing the indentation or changing tabs to spaces. Adding whitespace where there was none before, or removing a whitespace completely is still shown as a change. If **Ignore all whitespaces** is checked all whitespace-only changes are excluded.

e) Check the **Only record the merge** checkbox (optional).

If you are using merge tracking support and you want to mark a revision as having been merged, without actually doing the merge here, check the **Only record the merge** checkbox. You might want to do this for two possible reasons. You make the changes by hand, then mark the change as merged so that the merge tracking algorithm is aware of it. Or you might want to prevent a particular revision from being merged by marking it as already merged. This will prevent future merging.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you do a dry run of the merge operation in order to see what files are affected and how without modifying the working copy at all. This is very helpful in detecting where conflicts may occur.

8. Press the **Merge** button.
The merge operation is executed.
9. Optionally resolve the conflicts that were created by the merge operation.

After the merge operation is finished it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.

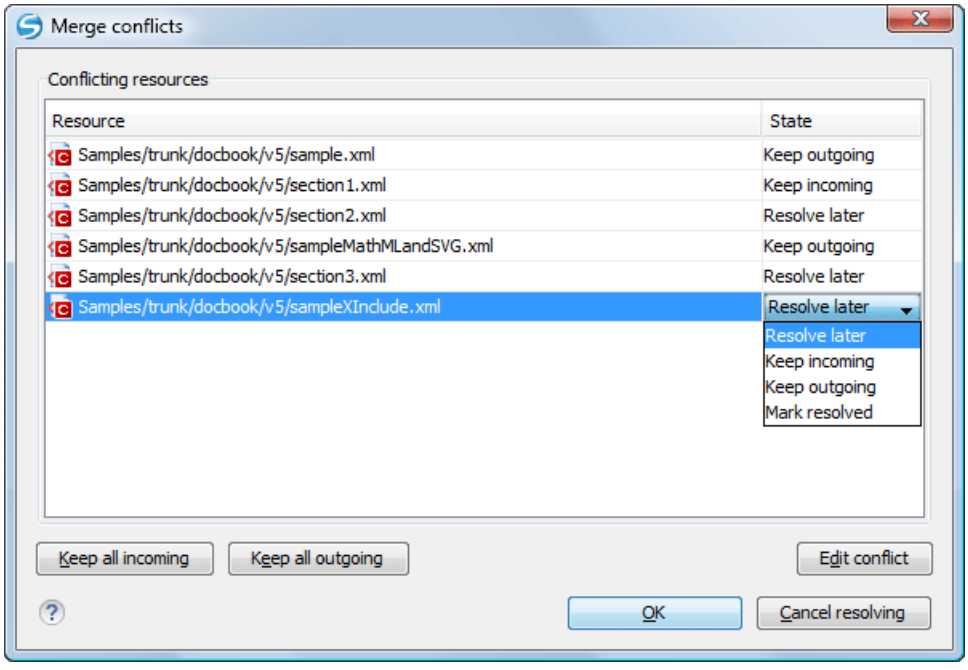


Figure 317: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used to leave the conflict as it is for manual resolving it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do that, use the **Edit conflict** button, which will bring to you a dialog presenting the conflicting resource's content for current working copy version and the one with the incoming modifications. After manually resolving the conflict, the resource will be marked as resolved.

When the merge is completed it's a good idea to look at the result of the merge in the specified working copy and see if it meets your expectations. Because merging is sometimes complicated, when there are major changes, *conflicts may appear*.

Merge Two Different Trees

This is a general case of the reintegrate method. You can consider the following example: calculate the changes necessary to get (from) the HEAD revision of the trunk (to) the HEAD revision of the branch, and apply those changes to my working copy (of the trunk). The result is that trunk will be identical with the branch.

Note: If the server does not support merge-tracking then this is the only way to merge a branch back to trunk.

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:

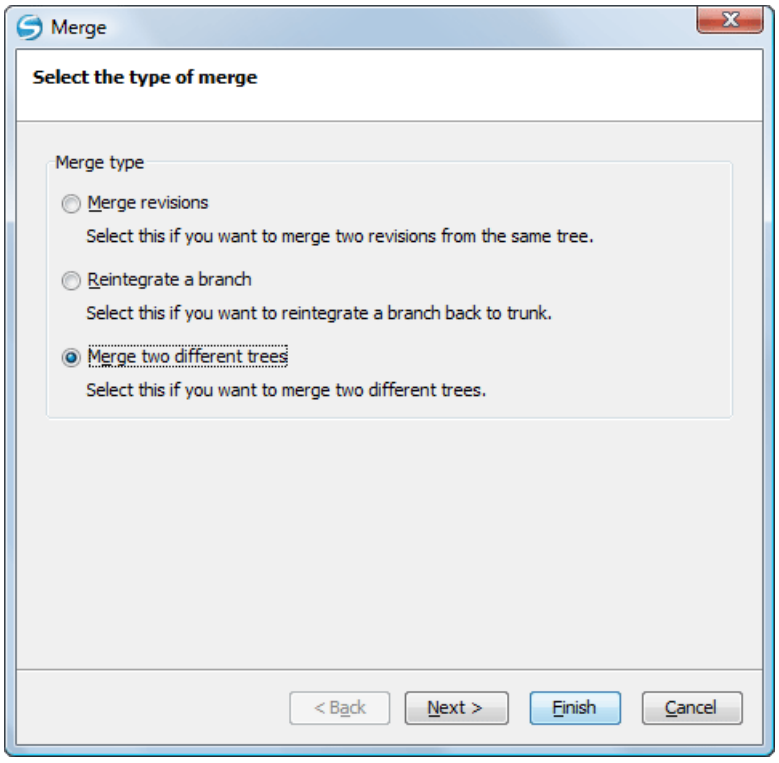


Figure 318: The Merge Wizard - The Merge Type

2. Select the option **Merge two different trees**.
3. Press the **Next** button.
   The second step of the **Merge** wizard is displayed:

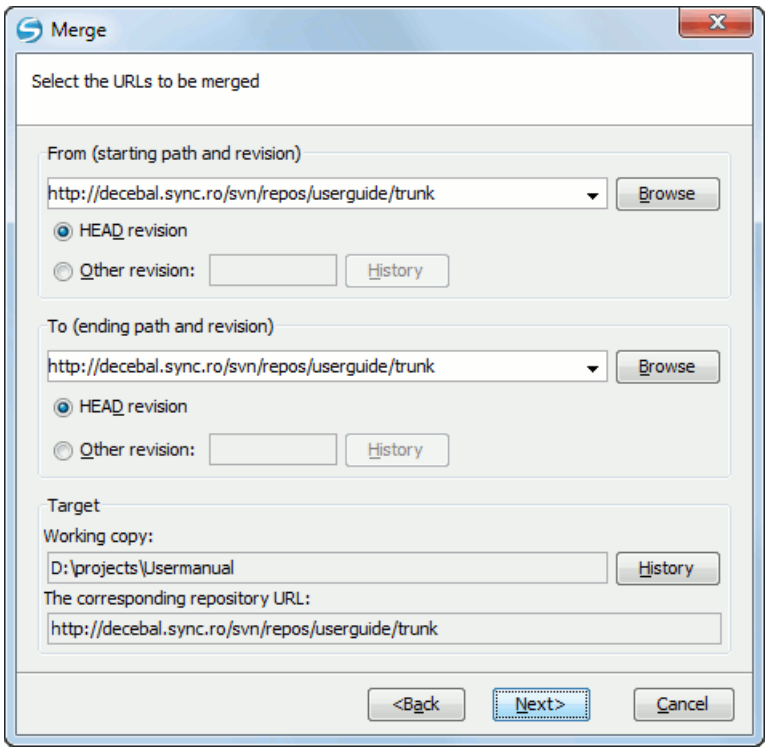


Figure 319: The Merge Wizard - Merge Two Different Trees

4. Specify the URL of the first tree in the **From** field.

   If you are using this method to merge a feature branch back to trunk, you need to start the merge wizard from within a working copy of trunk. In the **From** field enter the full folder URL of the trunk. This may sound wrong, but remember that the trunk is the start point to which you want to add the branch changes. In the **To** field enter the full folder URL of the feature branch.

   By default the start URL will be the URL of the selected file in the working copy. If you want to specify a different URL you should browse the repository and select a start URL and a revision.

   a) Select a URL in the **From** field.
   b) Select a revision of the repository from the **From** field.

      In the **Revision** field enter the last revision number at which the two trees were synchronized. If you are sure no-one else is making commits you can use the HEAD revision in both cases. If there is a chance that someone else may have made a commit since that synchronization, use the specific revision number to avoid losing more recent commits.

      By default the HEAD revision is selected. If you want a previous revision you have to select the **Other revision** option and press *the **History** button* to see a list of all revisions.

5. Specify the URL of the second tree in the **To** field.

   a) Select a URL in the **To** field.
   b) Select a revision of the repository from the **To** field.

      By default the HEAD revision is selected. If you want a previous revision you have to select the **Other revision** option and press *the **History** button* to see a list of all revisions.

6. Specify the target of the merge operation in the **Target** panel.

   The **Target** panel of the dialog reminds you the location of the target resource from the working copy where the merge result will be saved and its corresponding repository URL.

   a) Specify the working copy path in the **Working copy** field.
   b) Specify the repository URL corresponding to the working copy.

7. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:

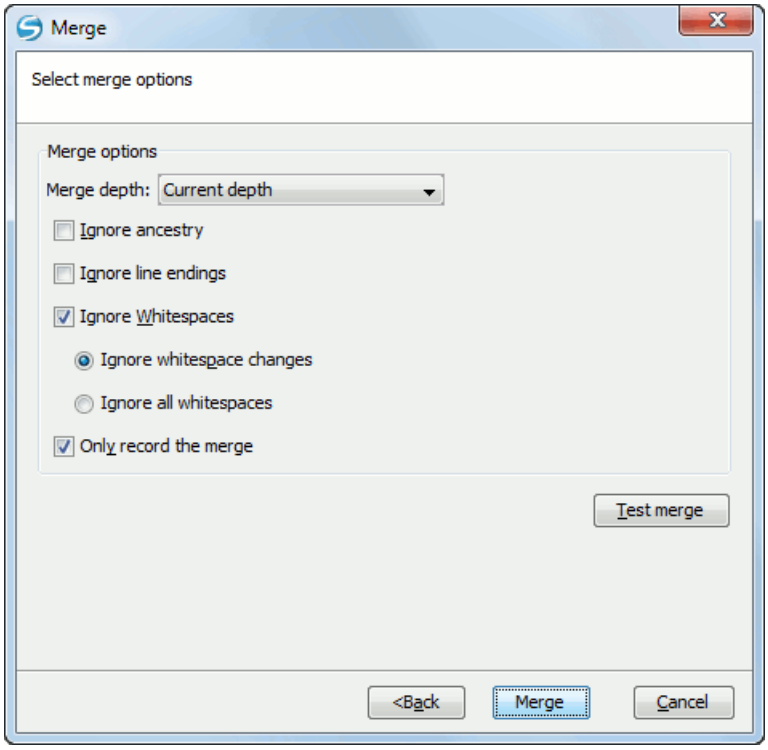


Figure 320: The Merge Wizard - Advanced Options

8. Set advanced options if necessary before starting the merge process.
   a) Set the depth of the merge operation in the **Merge depth** combo box.

      You can specify how far down into your working copy the merge should go by selecting one of the following values:

      - Current depth
      - Recursive (infinity)
      - Immediate children (immediates)
      - File children only (files)
      - This folder only (empty)

      The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

   b) Check the **Ignore ancestry** checkbox (optional).

      The **Ignore ancestry** checkbox allows a merge to be applied between a branch and the trunk or between two branches even if they do not share a common ancestry. Normally the branch and the trunk or the two branches that are merged must have a common ancestor revision in the same repository. In case the two merged trees were imported in the repository they are not related in the sense of a common ancestor tree and the merge operation is possible by ignoring the missing common ancestry of the two merged trees.

   c) Check the **Ignore line endings** checkbox (optional).

   d) Check the **Ignore Whitespaces** checkbox (optional).

      The **Ignore line endings** and **Ignore whitespaces** checkboxes allow you to specify how the line endings and whitespace changes should be handled. If they are checked the changes due only to the line endings and whitespaces are ignored. The default behavior is to treat all whitespace and line-end differences as real changes to be merged. **Ignore whitespace changes** excludes changes which are caused by a change in the amount or type of whitespace, for example changing the indentation or changing tabs to spaces. Adding whitespace where there was none before,

or removing a whitespace completely is still shown as a change. If **Ignore all whitespaces** is checked all whitespace-only changes are excluded.

e) Check the **Only record the merge** checkbox (optional).

If you are using merge tracking support and you want to mark a revision as having been merged, without actually doing the merge here, check the **Only record the merge** checkbox. You might want to do this for two possible reasons. You make the changes by hand, then mark the change as merged so that the merge tracking algorithm is aware of it. Or you might want to prevent a particular revision from being merged by marking it as already merged. This will prevent future merging.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you do a dry run of the merge operation in order to see what files are affected and how without modifying the working copy at all. This is very helpful in detecting where conflicts may occur.

9. Press the **Merge** button.
The merge operation is executed.
10. Optionally resolve the conflicts that were created by the merge operation.

After the merge operation is finished it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.

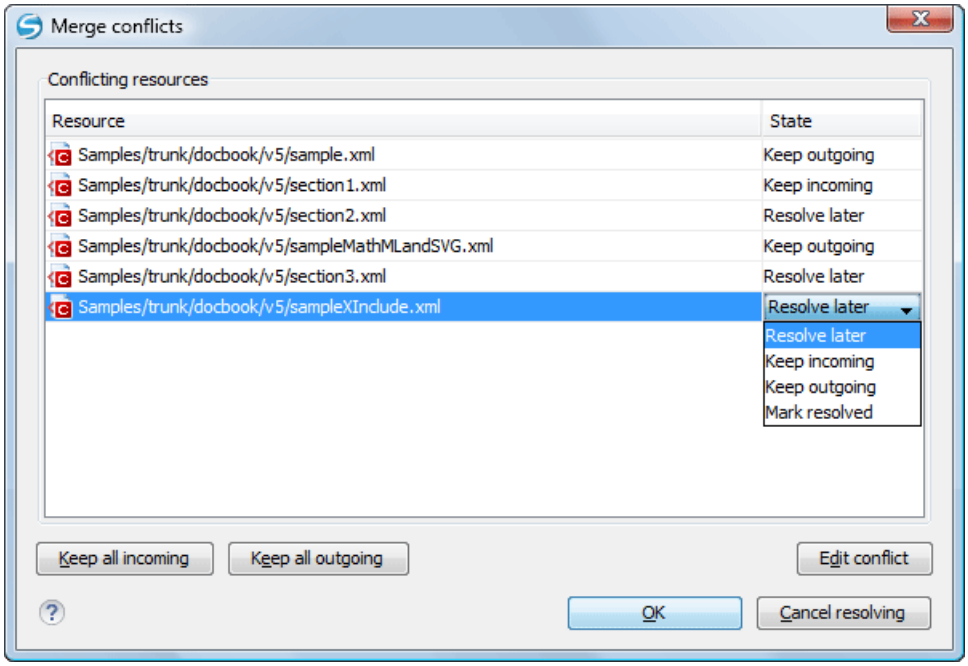


Figure 321: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used to leave the conflict as it is for manual resolving it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do that, use the **Edit conflict** button, which will bring to you a dialog presenting the conflicting resource's content for current working copy version and the one with the incoming modifications. After manually resolving the conflict, the resource will be marked as resolved.

Merge Options

1. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:

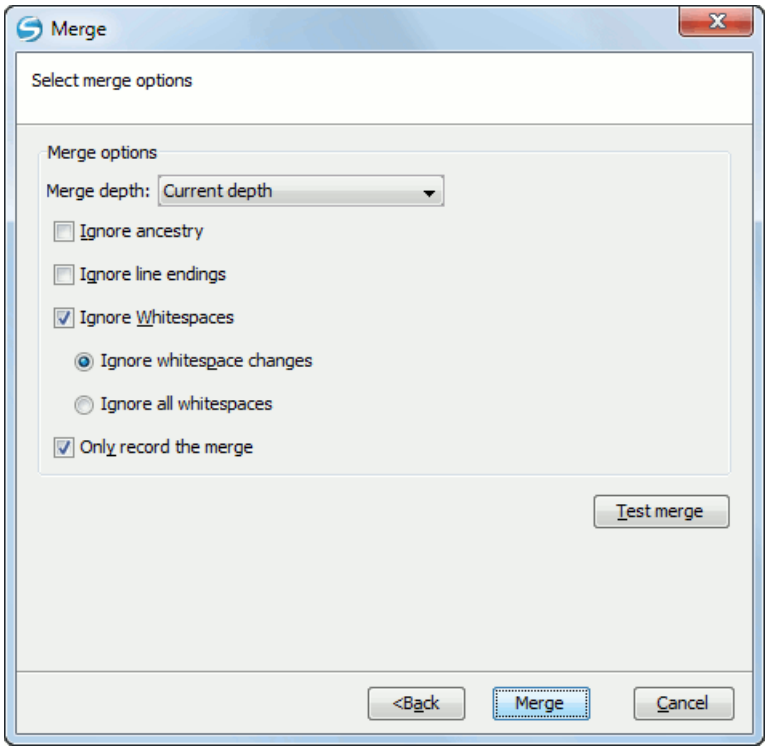


Figure 322: The Merge Wizard - Advanced Options

2. Set advanced options if necessary before starting the merge process.
   a) Set the depth of the merge operation in the **Merge depth** combo box.

      You can specify how far down into your working copy the merge should go by selecting one of the following values:

      - Current depth
      - Recursive (infinity)
      - Immediate children (immediates)
      - File children only (files)
      - This folder only (empty)

      The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

   b) Check the **Ignore ancestry** checkbox (optional).

      The **Ignore ancestry** checkbox allows a merge to be applied between a branch and the trunk or between two branches even if they do not share a common ancestry. Normally the branch and the trunk or the two branches that are merged must have a common ancestor revision in the same repository. In case the two merged trees were imported in the repository they are not related in the sense of a common ancestor tree and the merge operation is possible by ignoring the missing common ancestry of the two merged trees.

   c) Check the **Ignore line endings** checkbox (optional).

   d) Check the **Ignore Whitespaces** checkbox (optional).

      The **Ignore line endings** and **Ignore whitespaces** checkboxes allow you to specify how the line endings and whitespace changes should be handled. If they are checked the changes due only to the line endings and whitespaces are ignored. The default behavior is to treat all whitespace and line-end differences as real changes to be merged.

Ignore whitespace changes excludes changes which are caused by a change in the amount or type of whitespace, for example changing the indentation or changing tabs to spaces. Adding whitespace where there was none before, or removing a whitespace completely is still shown as a change. If **Ignore all whitespaces** is checked all whitespace-only changes are excluded.

e) Check the **Only record the merge** checkbox (optional).

If you are using merge tracking support and you want to mark a revision as having been merged, without actually doing the merge here, check the **Only record the merge** checkbox. You might want to do this for two possible reasons. You make the changes by hand, then mark the change as merged so that the merge tracking algorithm is aware of it. Or you might want to prevent a particular revision from being merged by marking it as already merged. This will prevent future merging.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you do a dry run of the merge operation in order to see what files are affected and how without modifying the working copy at all. This is very helpful in detecting where conflicts may occur.

3. Press the **Merge** button.
The merge operation is executed.

When the merge is completed it's a good idea to look at the result of the merge in the specified working copy and see if it meets your expectations. Because merging is sometimes complicated, when there are major changes, *conflicts may appear*.

Merge Branches Task - First Step

Go to menu **Tools** > **Merge ...**
The **Merge** wizard is opened:

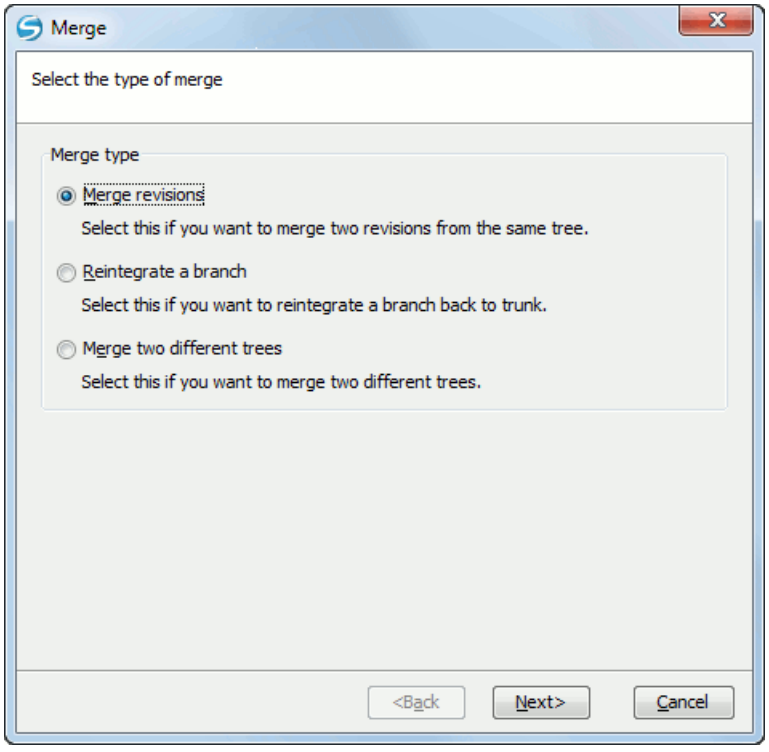


Figure 323: The Merge Wizard - The Merge Type

Resolve Merge Conflicts

Optionally resolve the conflicts that were created by the merge operation.

After the merge operation is finished it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are

such conflicts, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.

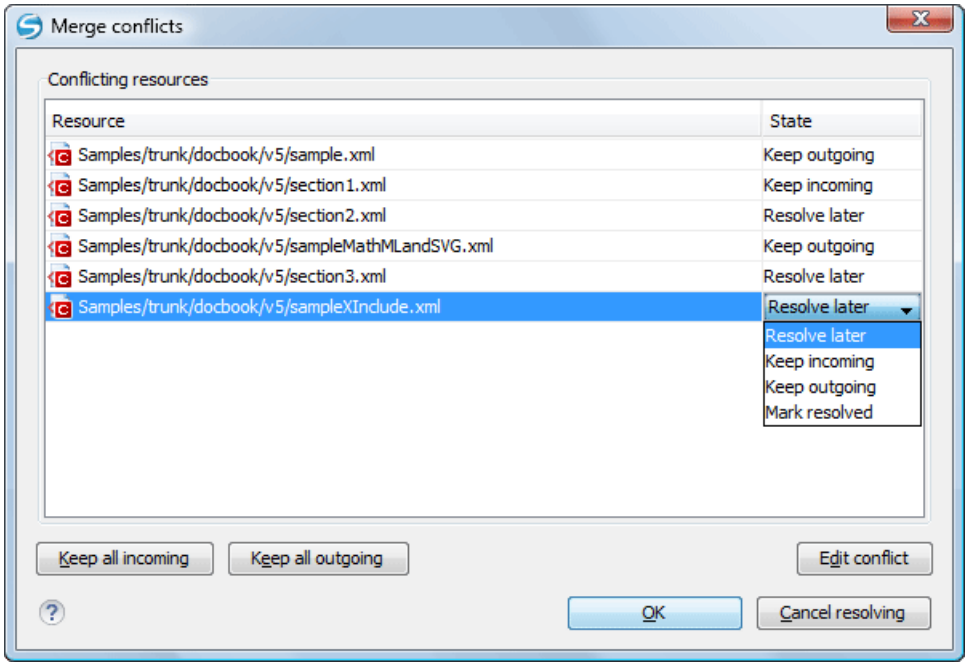


Figure 324: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used to leave the conflict as it is for manual resolving it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do that, use the **Edit conflict** button, which will bring to you a dialog presenting the conflicting resource's content for current working copy version and the one with the incoming modifications. After manually resolving the conflict, the resource will be marked as resolved.

Switch the Repository Location

The **Switch** action is useful when the repository location of a working copy or only of a versioned item of the working copy must be changed within the same repository. The action is available on the **Tools** menu when a versioned resource is selected in the current working copy that is displayed in *the **Working Copy** view*.

Relocate a Working Copy

Sometimes the base URL of the repository is changed after a working copy is checked out from that URL, for example the repository itself is relocated to a different server. In such cases you do not have to check out again a working copy from the new repository location. It is easier to change the base URL of the root folder of the working copy to the new URL of the repository. This action is called **Relocate** and is available on the **Tools** menu when the selected item in *the **Working Copy** view* is a versioned folder.

If the selected item is not the root folder of the working copy then the effect is the same as for *the **Switch** action* applied on the same selected item.

Patches

This section explains how to work with patches in Syncro SVN Client.

What Is a Patch

Let's suppose you are working to a set of XML files, that you distribute to other people. From time to time you are tagging the project and distribute the releases. If you continue working for a period correcting problems, you may find yourself in the situations to notify your users that you have corrected a problem. In this case you may prefer to distribute them a patch, a collection of differences that applied over the last distribution would correct the problem. The SVN client creates the patch in *the Unified Diff format*.

Creating a patch in Subversion implies the access to two states (revisions) of a project:

- the current working copy and a revision from the repository - if you have not committed yet your current working copy and prefer not to do it, you create a patch between the current working copy and a revision from the repository;
- two repository revisions - if both states are two revisions already committed to the repository.

Create a Patch Between Working Copy and Repository Revision

When the changes that must be included in a patch were not committed to the repository the patch should be created with the differences between the selected item(s) of the working copy and a repository revision. The steps are the following:

1. Go to menu **Tools** > **Create patch** .
   This opens the **Create patch** wizard:

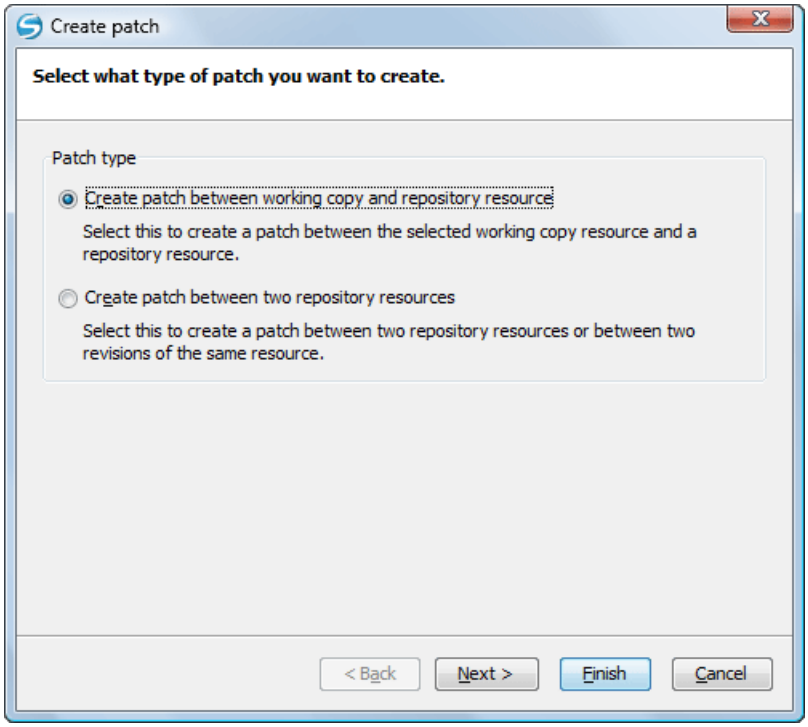


Figure 325: The Create Patch Wizard - Patch Type

2. Select the first option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:

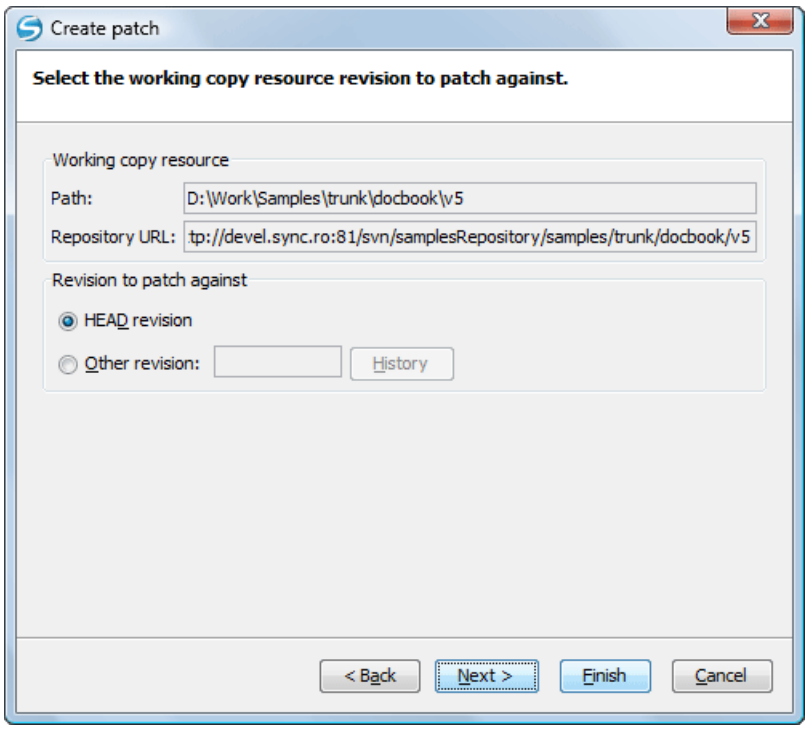


Figure 326: The Create Patch Wizard - Step 2

4. Specify the working copy resource.
   a) Specify the local file path of the working copy resource.
   b) Specify the repository URL corresponding to the working copy resource.
5. Select the repository revision.

   You have the option of choosing between the HEAD revision and a specific revision number. For the second option you should press *the **History** button* to display a list of the repository revisions.
6. Press the **Next** button.
   The next step of the wizard is displayed:

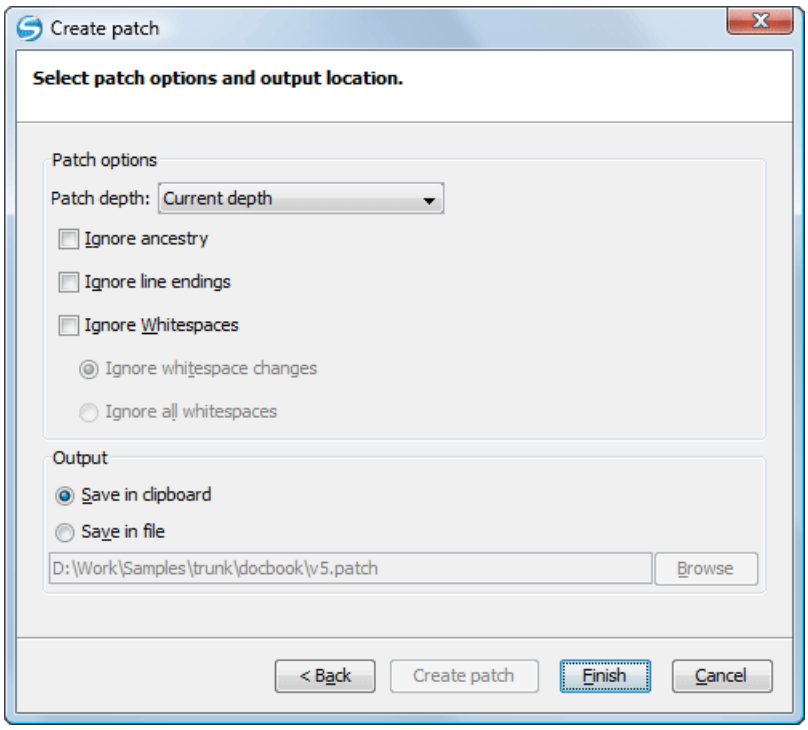


Figure 327: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

   In case of a file resource the depth is always zero. In case of a folder resource the depth has one of the following values:

   - **Current depth** - The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
   - **Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.
   - **Immediate children (immediates)** - The patch is created only on the child files and folders without going in subfolders.
   - **File children only (files)** - The patch is created only on the child files.
   - **This folder only (empty)** - The patch is created only on the selected folder (that is no child file or folder is included in the patch).

8. Select the **Ignore ancestry** checkbox (optional).

   If checked, the SVN ancestry that exists when the two URLs have a common SVN history is ignored.

9. Select the **Ignore line endings** checkbox (optional).

   If checked, the differences in line endings are ignored when the patch is created.

10. Select the **Ignore whitespaces** checkbox (optional).

    If checked, the differences in whitespaces are ignored when the patch is created.

11. Select the output location.

    - **Save in clipboard** - The patch will be created and saved in clipboard. This is useful when you do not want to save the patch in a file on disk.
    - **Save in file** - The patch will be created and saved in the specified file.

12. Press the **Next** button.

    This will go to the final step of the wizard:

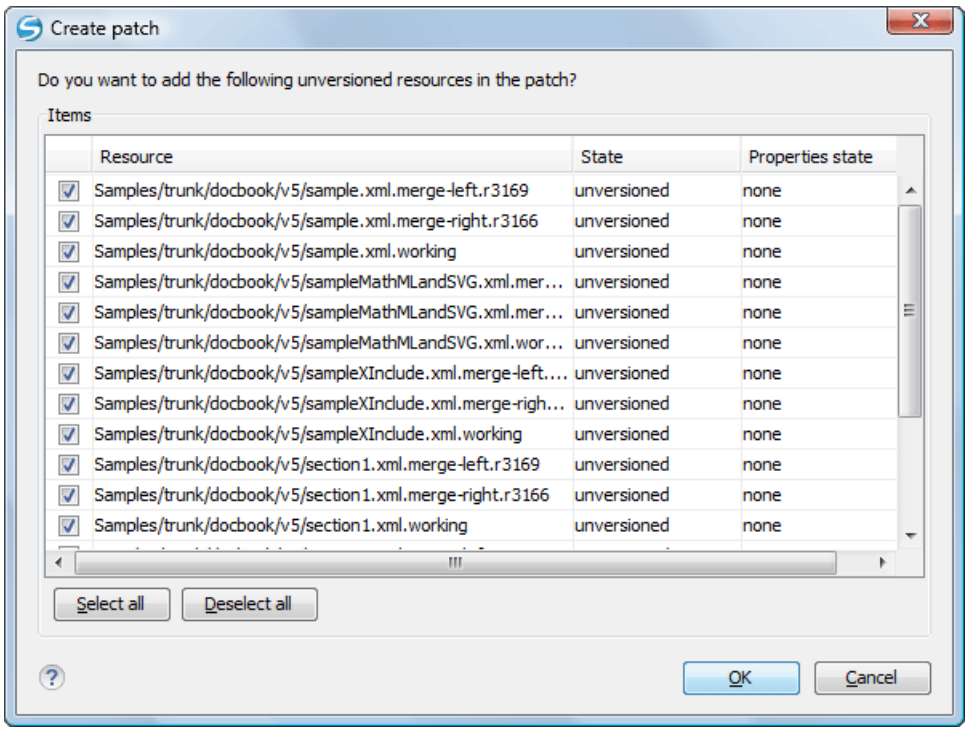


Figure 328: The Patch Wizard - Add Unversioned Resources

13. Select the unversioned files that will be included in the patch.

If the patch is applied on a folder of the working copy and that folder contains unversioned files this step of the wizard offers the option of selecting the ones that will be included in the patch.

14. Press the OK button.
The patch is created by applying all the specified options.

Create a Patch Between Two Repository Revisions

When a patch must include the differences between two repository revisions, in the same repository or in two different repositories, the steps for creating the patch are the following:

1. Go to menu **Tools** > **Create patch** .
This opens the **Create patch** wizard:

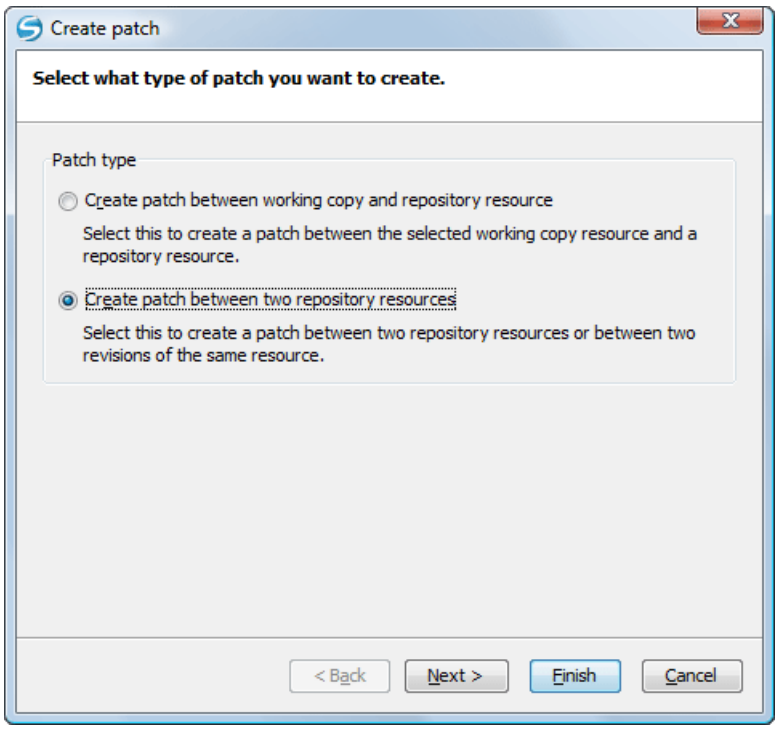


Figure 329: The Create Patch Wizard - Patch Type

2. Select the second option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:

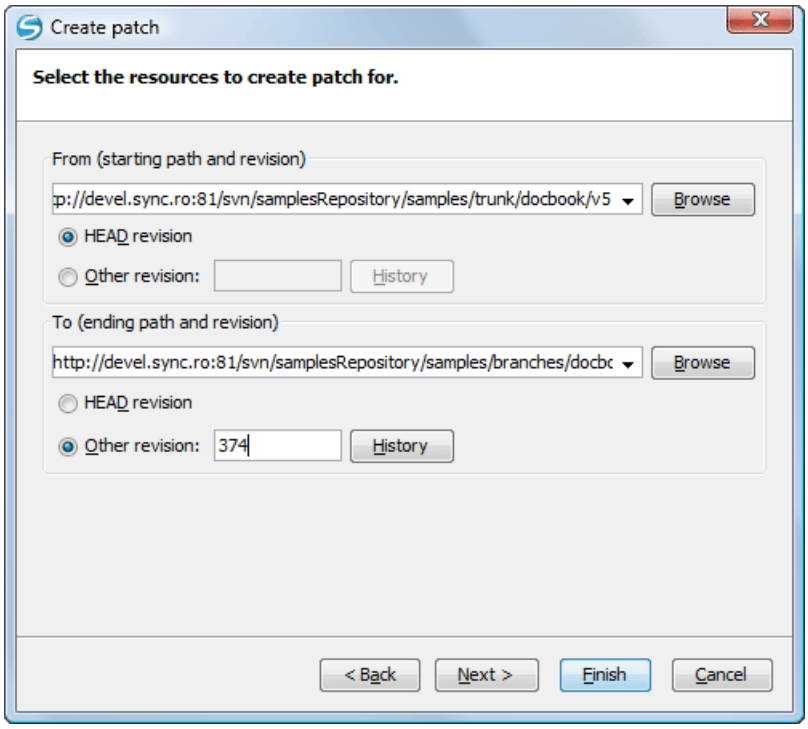


Figure 330: The Create Patch Wizard - Step 2

4. Select the first repository revision in the **From** panel.
5. Select the second repository revision in the **To** panel.

For both revisions you have the option of choosing between the HEAD revision and a specific revision number. For a specific number you should press *the **History** button* to display a list of the repository revisions.

6. Press the **Next** button.
   The next step of the wizard is displayed:

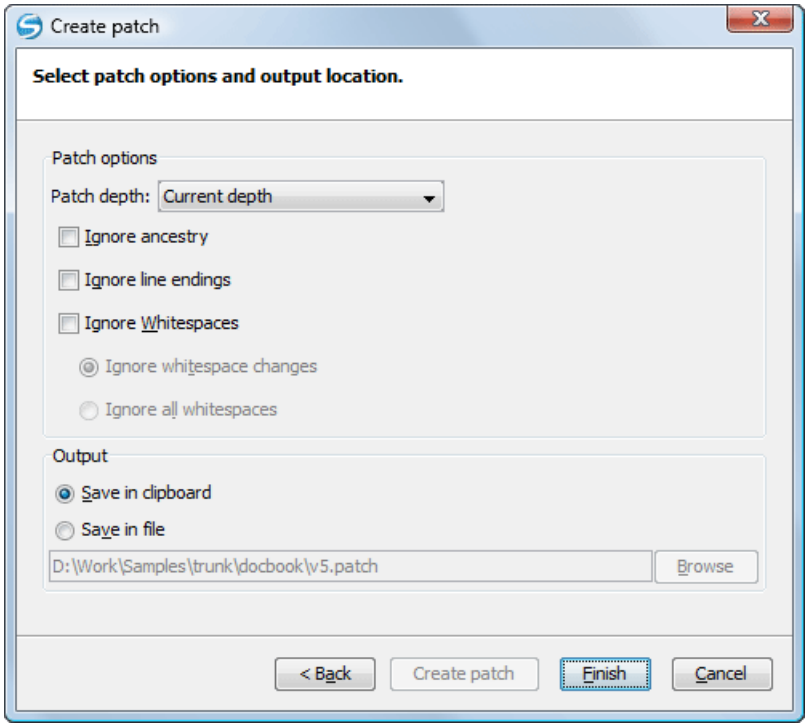


Figure 331: The Create Patch Wizard - Options

7. Select the depth of the patch operation.
   In case of a file resource the depth is always zero. In case of a folder resource the depth has one of the following values:
   - **Current depth** - The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
   - **Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.
   - **Immediate children (immediates)** - The patch is created only on the child files and folders without going in subfolders.
   - **File children only (files)** - The patch is created only on the child files.
   - **This folder only (empty)** - The patch is created only on the selected folder (that is no child file or folder is included in the patch).
8. Select the **Ignore ancestry** checkbox (optional).
   If checked, the SVN ancestry that exists when the two URLs have a common SVN history is ignored.
9. Select the **Ignore line endings** checkbox (optional).
   If checked, the differences in line endings are ignored when the patch is created.
10. Select the **Ignore whitespaces** checkbox (optional).
    If checked, the differences in whitespaces are ignored when the patch is created.
11. Select the output location.
    - **Save in clipboard** - The patch will be created and saved in clipboard. This is useful when you do not want to save the patch in a file on disk.
    - **Save in file** - The patch will be created and saved in the specified file.

12. Press the **Next** button.
    This will go to the final step of the wizard:

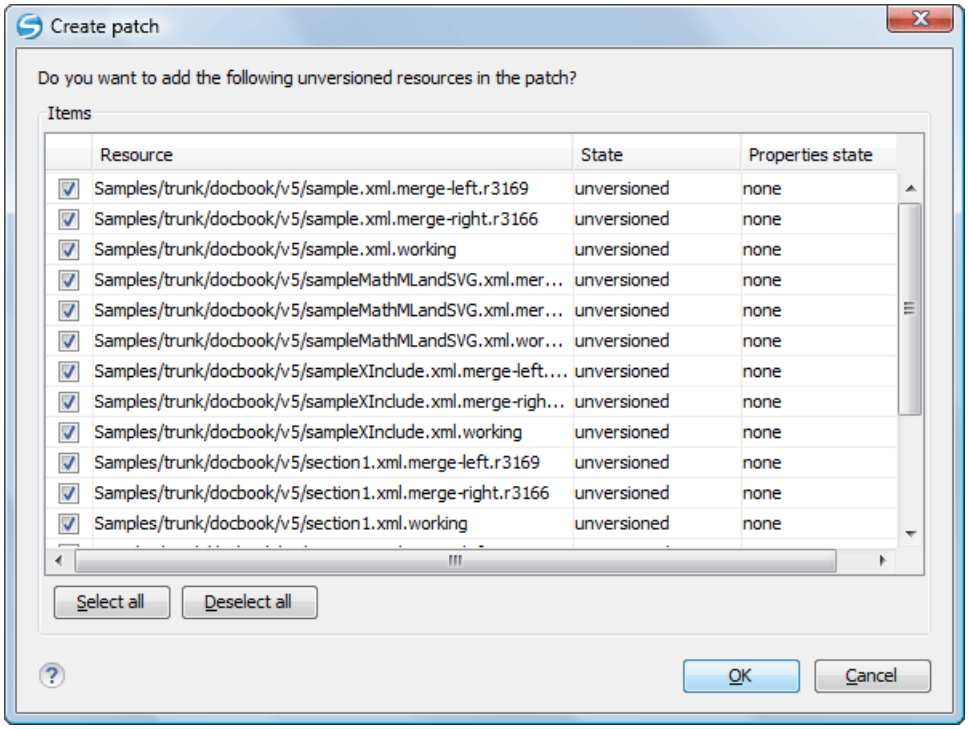


Figure 332: The Patch Wizard - Add Unversioned Resources

13. Select the unversioned files that will be included in the patch.

    If the patch is applied on a folder of the working copy and that folder contains unversioned files this step of the wizard offers the option of selecting the ones that will be included in the patch.

14. Press the OK button.
    The patch is created by applying all the specified options.

Working with Repositories

This section explains how to locate and browse SVN repositories in Syncro SVN Client.

Importing Resources Into a Repository

This is the process of taking a project and importing it into a repository so that it can be managed by a Subversion server. If you have already been using Subversion and you have an existing working copy you want to use, then you will likely want to follow the procedure for *using an existing working copy*.

This process is started from menu **Repository** > **Import** > **Import Folder Content** . The same action is available in the **Repositories** view contextual menu. A dialog will ask you to select a directory that will be imported into the selected repository location. The complete directory tree will be imported into the repository including all files. The name of the imported folder will not appear in the repository, but only the contents of the folder will.

Exporting Resources From a Repository

This is the process of taking a resource from the repository and saving it locally in a clean form, with no version control information. This is very useful when you need a clean build for an installation kit.

The export dialog is very similar to the check out dialog:

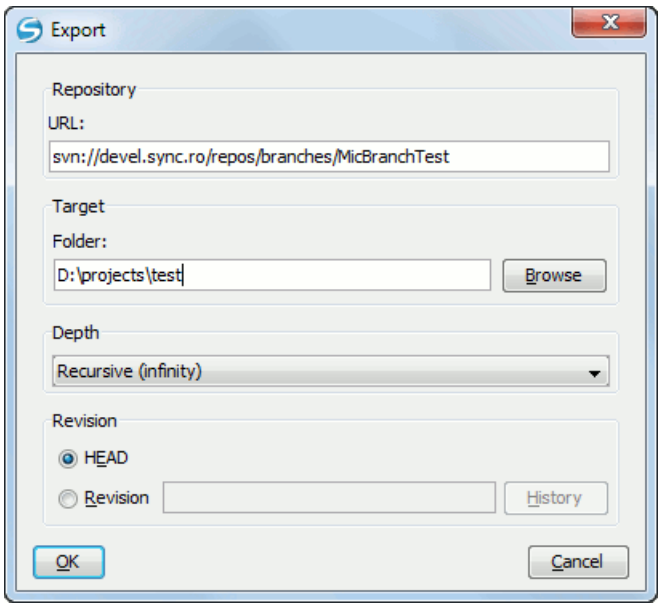


Figure 333: Export from Repository

You can choose the target directory from the file system by pressing the **Browse** button. If you need to export a specific revision, you can select the **Revision** radio button and then click on the **History** button and choose a revision from the new dialog. Or you could simply type the revision number in the corresponding text field.

Please note that the content of the selected directory from the repository and not the directory itself will be exported to the file system.

Copy / Move / Delete Resources From a Repository

Once you have a location defined in the ***Repositories*** *view* you can execute commands like copy, move and delete directly on the repository. The commands correspond to the following actions in the contextual menu:

- The **Copy/Move** action allows you to copy and move individual or multiple resources. After invoking the action the **Copy/Move** dialog will pop up. The dialog displays the path of the resource that is copied and the tree structure of the repository allowing you to choose the destination directory. The path of this target directory will be presented in the text field **URL**. If you choose to copy a single resource then an additional checkbox called **Change name** and a text field allow you to choose the new name of the copied/moved resource.

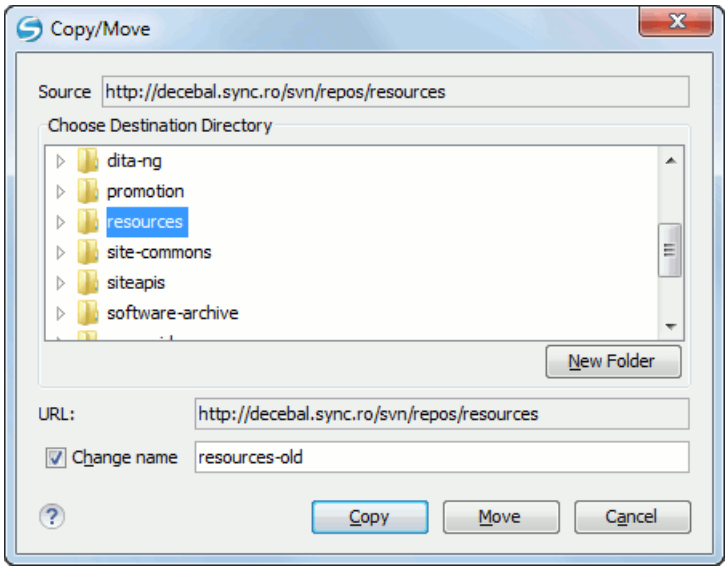


Figure 334: Copy/Move Resources on a Repository

- Another useful action is **Delete**. This action allows you to delete resources directly from the repository. After choosing the action from the ***Repositories*** *view contextual menu* a confirmation dialog will be displayed.

All three actions are commit operations and you will be prompted with the **Commit message** dialog.

Sparse Checkout

Sometimes you need to check out only certain parts of a directory tree. For this you can checkout the top folder (*the action* ***Check Out*** *from the* ***Repositories*** *view*) and then update recursively only the needed directories (*the action* ***Update*** *from the* ***Working Copy*** *view*). Each directory now understands the notion of depth which has four possible values:

- **Recursive (infinity)** - Updates all descendant folders and files recursively.
- **Immediate children** - Updates the folder including direct child folders and files but does not populate the child folders.
- **File children only (files)** - Updates the directory including only child files without the child folders.
- **This folder only (empty)** - Updates only the selected directory without updating any children.

For some operations you can use as depth the current depth of the resource from working copy (the value **Current depth**). This is the depth of directories from the working copy and it can have one of the values defined above. This is the depth value defined in a previous checkout or update operation.

The sparse checked out directories are marked in *the* ***Working Copy*** *view* with a marker corresponding to the depth value as follows:

- **Recursive (infinity)** - This is the default value and it is has no mark.
- **Immediate children (immediates)** - The directory is marked with a purple bubble  in the top left corner.
- **File children only (files)** - The directory is marked with a blue bubble  in the top left corner.
- **This folder only (empty)** - The directory is marked with a gray bubble  in the top left corner.

The depth information is also presented in the **Information** view and the tool tip displayed when hovering over the directory in the **Working Copy** view.

This feature requires the SVN client to support the SVN format 1.5 or above and will work most efficiently if the server is 1.5 or above. The client will work also with a 1.4 server or lower but will be less efficient.

Syncro SVN Client Views

The main working area occupies the center of the application window, which contains the three most important views:

- *Working Copy View*
- *History View*
- *Console View*

The other views that support the main working area are also presented in this section.

Repositories View

The **Repositories** view allows you to define and manage Subversion repository locations and browse repositories. Repository files and folders are presented in a tree view with the repository locations at the first level, where each location represents a connection to a specific repository. More information about each resource is displayed in a tabular form:

- **Date** - Date when the resource was last modified;
- **Revision** - The revision number at which the resource was last time modified;
- **Author** - Name of the person who made the last modification on the resource;
- **Size** - Resource size on disk;

- **Type** - Contains the resource type or file extension.

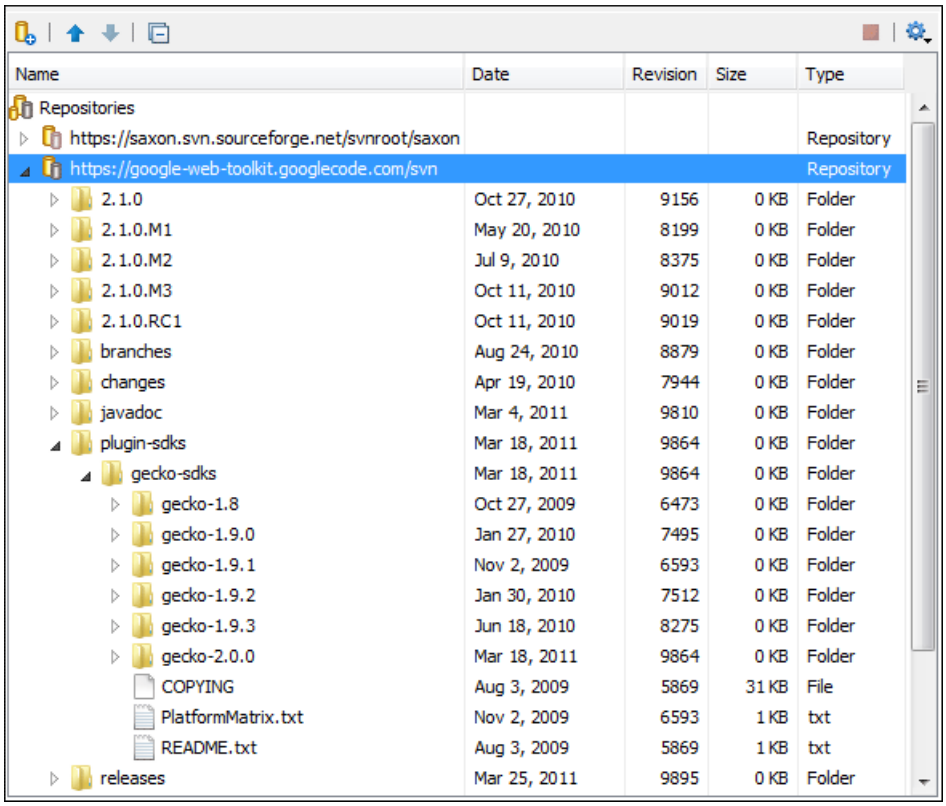


Figure 335: Repositories View

Toolbar

The **Repositories** view's toolbar contains the following buttons:

- **New Repository Location** - Allows you to enter a new repository location by means of the **Add SVN Repository** dialog.
- **Move Up** - Move the selected repository up one position in the list of repositories in the **Repositories** view.
- **Move Down** - Move the selected repository down one position in the list of repositories in the **Repositories** view.
- **Collapse all** - Collapses all repository trees.
- **Stop** - Stops the current repository browsing operation executed when a repository node is expanded. This is useful when the operation takes too long or the server is not responding.
- **Settings** - Allows you to configure the resource table appearance.

Contextual Menu Actions

The **Repositories** view contextual menu contains different actions depending on the selected item. If a repository location is selected, the following management actions are available :

- **New Repository Location (Ctrl + Alt + N)** - Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.

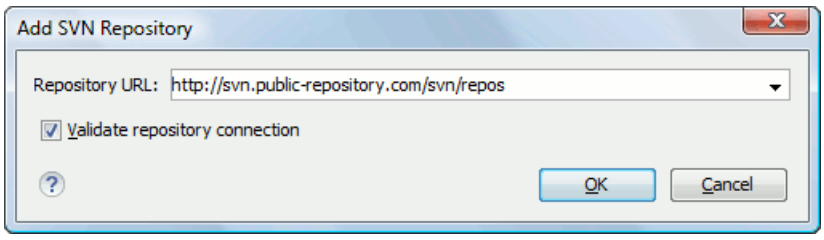


Figure 336: Add SVN Repository

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

- **Edit Repository Location (Ctrl + Alt + E)** - Context-dependent action that allows you to edit the selected repository location by means of the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.
- **Change the Revision to Browse (Ctrl + Alt + Shift + B)** - Context-dependent action that allows you to change the selected repository revision by means of the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.
- **Remove Repository Location (Ctrl + Alt + Shift + R)** - Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.

The following action are common to all repository resources:

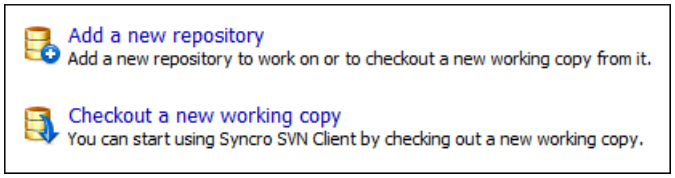
- **Refresh** - Refreshes the resource selected in the **Repositories** view.
- **Check Out (Ctrl + Alt + Shift + C)** - Allows you to copy resources from a repository into your local file system. To use this operation, you must select a repository root location or a folder from a repository, but never a file. If you don't select anything, you can specify an URL to a folder resource from a repository in the **Check Out** dialog that appears when performing this operation. To read more about this operation, see the section *Check out a working copy*.
- **Export** - Exports a folder from the repository to the local file system.
- **Import** sub-menu:
  - **Import Folder Content (Ctrl + Alt + Shift + M)** - Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.
  - **Import File(s) (Ctrl + Alt + I)** - Imports the files selected from the files system into the selected folder from the repository.
- **New Folder** - Allows you to create a new folder in the selected repository path.
- **Open** - Opens the selected file in the Editor view in read-only mode.
- **Open with** - Displays the *Open with...* dialog to specify the editor in which the selected file will be opened. In case multiple files are selected, only external applications can be used to open the files.
- **Copy/Move to (Ctrl + M)** - Copies or moves to a specified location the selected resource.
- **Rename (F2)** - Renames the selected resource.
- **Delete (Delete)** - Deletes the selected resource.
- **Copy URL Location (Ctrl + Alt + U)** - Copies to clipboard the URL location of the selected resource.
- **Branch/Tag** - Allows you to create a branch or a tag from the selected folder in the repository. To read more about how to create a branch/tag, please see the *Creation and management of Branches/Tags* section.
- **Show History (Ctrl + H)** - Displays the history of the selected resource. At the start of the operation you can set filtering options.
- **Show Annotation (Ctrl + Shift + A)** - Complex action that does the following operations:

- opens the selected resource in the **Annotations** editor;
- displays corresponding annotations list in the **Annotations** view;
- displays the history of the selected resource.

- **Revision Graph (Ctrl + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.
- **Show SVN Properties (Ctrl + Shift + P)** - Brings up the *Properties view* displaying the SVN properties for the selected resource. This view does not allow adding, editing or removing SVN properties of a repository resource. These operations are allowed only for working copy resources.
- **File Information (Ctrl + I)** - provides additional information for the selected resource. For more details please see the section *Information view*.

Assistant Actions

When there is no repository configured, the **Repositories** view mode lists the following two actions:



Working Copy View

The **Working Copy** view allows you to manage the content of an SVN working copy.

The toolbar contains the list of defined working copies, a set of view modes that allow you to filter the content of the working copy based on the resource status (like incoming or outgoing changes), and a **Settings** menu.

If you click any of the view modes (All Files, Modified, Incoming, Outgoing, Conflicts), the information displayed changes as follows:

- **All Files** - Resources (files and folders) are presented in a hierarchical structure with the root of the tree representing the location of the working copy on the file system. Each resource has an icon representation which describes the type of resource and also depicts the state of that resource with a *small overlay icon*.

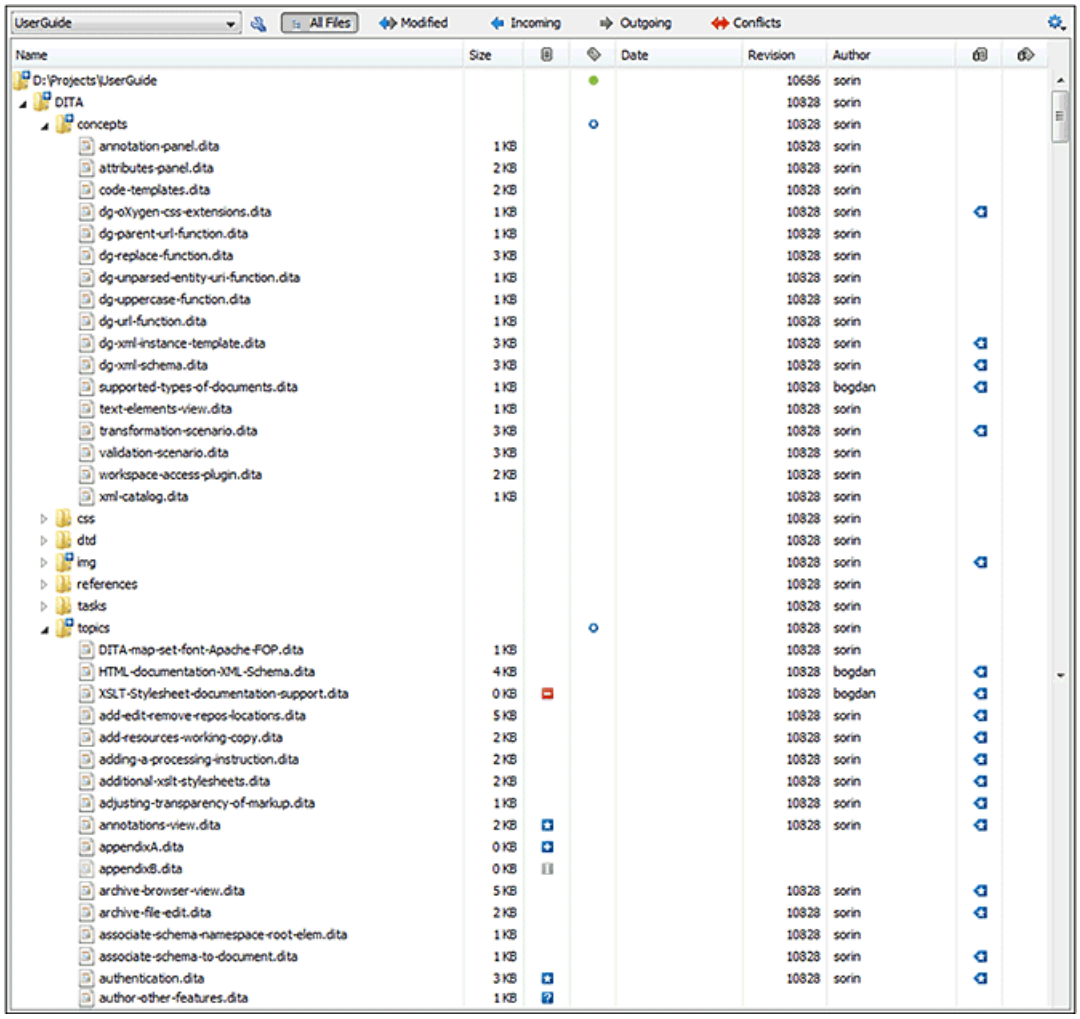


Figure 337: Working Copy View - All Files View Mode

- **Modified** - The resource tree presents resources modified locally (including those with conflicting content) and remotely. Decorator icons are used to differentiate between various resource states:
  - - incoming modification from repository:
    - - file content modified remotely;
    - - new file added remotely;
    - - file deleted remotely;
  - - outgoing modification to repository:
    - - file content modified locally;
    - - new file added locally;
    - - file deleted locally;
  - - pseudo-conflict state - a resource being locally and remotely modified at the same time;
  - - real conflict state - a resource that had both incoming and outgoing changes and not all the differences could be merged automatically by the update operation (manually editing the local file is necessary for resolving the conflict).

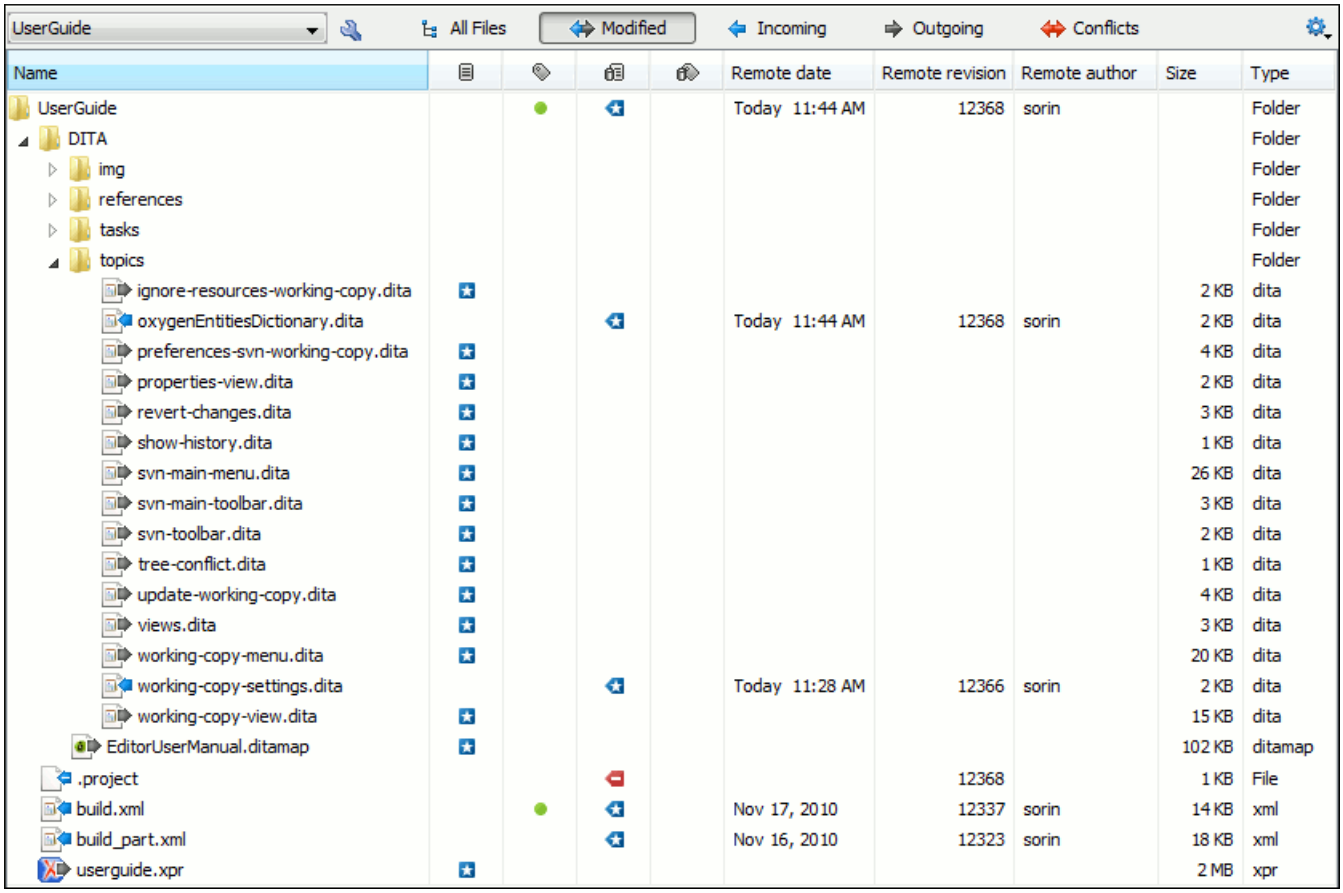


Figure 338: Working Copy View - Modified View Mode

- **Incoming** - The resource tree presents only incoming changes.
- **Outgoing** - The resource tree presents only outgoing changes.
- **Conflicts** - The resource tree presents only conflicting changes (real conflicts and pseudo-conflicts).

The following columns provide information about the resources:

- **Name** - Resource name. Resource icons can have the following decorator icons:
  - Additional status information:
    - **Propagated modification marker** - A folder marked with this icon indicates that the folder itself presents some changes (like modified properties) or a child resource has been modified.
    - **External** - This indicates a mapping of a local directory to the URL of a versioned resource. It is declared with a *svn:externals* property in the parent folder.
    - **Switched** - This indicates a resource that has been switched from the initial repository location to a new location within the same repository. The resource goes to this state as a result of *the Switch action* executed from the contextual menu of the Working Copy view.
    - *Grayed* - A resource with a grayed icon but no overlaid icon is an ignored resource. It is obtained with the **Add to svn:ignore** action.
  - Current SVN depth of a folder:
    - **Immediate children (immediates)** (a variant of *sparse checkout*) - The directory contains only direct file and folder children. Child folders ignore their content.

- **File children only (files)** (a variant of *sparse checkout*) - The directory contains only direct file children, disregarding any child folders.
- **This folder only (empty)** (a variant of *sparse checkout*) - The directory will discard any child resource.

Note: Any folder unmarked with one of the depth icons, has recursive depth (infinity) set by default (presents all levels of child resources).

- **Size** - Resource size on disk;
- **Local file status** - Shows the changes of working copy resources that were not committed to the repository yet. The following icons are used to mark resource status:
  - - Resource is not under version control.
  - - Resource is being ignored because it is not under version control and its name matches a file name pattern defined in one of the following places:
    - *global-ignores* section in the SVN client-side *'config' file*;
    - ***Application global ignores** option* of Syncro SVN Client;
    - the value of a *svn:ignore property* set on the parent folder of the resource being ignored.
  - - Marks a newly created resource, scheduled for addition to the version control system.
  - - Marks a resource scheduled for addition, created by copying a resource already under version control and inheriting all its SVN history.
  - - The content of the resource has been modified.
  - - Resource has been replaced in your working copy (the file was scheduled for deletion, and then a new file with the same name was scheduled for addition in its place).
  - - Resource is scheduled for deletion.
  - - The resource is incomplete (as a result of an interrupted *checkout* or *update* operation).
  - - The resource is missing because it was moved or deleted without using a SVN aware application.
  - - The contents of the resource is in *real conflict state*.
  - - Resource is in tree conflict state after an update operation because:
    - Resource was locally modified and incoming deleted from repository;
    - Resource was locally scheduled for deletion and incoming modified.
  - - Resource is obstructed (versioned as one kind of object: file, directory or symbolic link, but has been replaced outside Syncro SVN Client by a different kind of object).
- **Local properties status** - Marks the resources that have SVN properties, with the following possible states:
  - - The resource has SVN properties set.
  - - The resource properties have been modified.
  - - Properties for this resource are in *real conflict* with property updates received from the repository.
- **Date** - Date when the resource was last time modified.
- **Revision** - The revision number at which the resource was last time modified.
- **Author** - Name of the person who made the last modification on the resource.
- **Remote file status** - Shows changes of resources recently modified in the repository. The following icons are used to mark incoming resource status:
  - - Resource is newly added in repository.
  - - The content of the resource has been modified in repository.
  - - Resource was replaced in repository.
  - - Resource was deleted from repository.

- **Remote properties status** - Resources marked with the icon have incoming modified properties from the repository.
- **Remote date** - Date of the resource's latest modification committed on the repository.
- **Remote revision** - Revision number of the resource's latest committed modification.
- **Remote author** - Name of the author who committed the latest modification on the repository.
- **Type** - Contains the resource type or file extension.

Note: The working copy table allows you to show or hide any of its columns and also to sort its contents by any of the displayed columns. The table header provides a contextual menu which allows you to customize the displayed information.

The toolbar allows you to switch between two working copies:

- Drop down list - Contains all the working copies Syncro SVN Client is aware of. When you select another working copy from the list, the newly selected working copy content will be scanned and displayed in the **Working Copy** view.
- **Add/Remove Working Copy** - opens the **Working copies list** dialog which displays the working copies Syncro SVN Client is aware of. In this dialog you can add existing working copies or remove the no longer needed ones. If you try to add a folder which is not a valid Subversion working copy, a warning dialog will inform you that the selected directory is not under version control. Please note that removing a working copy from this dialog will NOT remove it from your file system; you will have to do that manually.

Working Copy Settings

The **Settings** button from the toolbar of the **Working Copy** view provides the following actions:

- **Show ignored files** - When selected, the application displays the ignored resource inside the **All Files** mode.
- **Show deleted files** - Allows displaying or hiding the deleted resource inside the **All Files** mode. All other modes always display deleted resources, disregarding this action.
- **Tree** / **Compressed** / **Flat** - Affect the way the information is displayed inside the **Modified**, **Incoming**, **Outgoing**, and **Conflicts** view modes.
- **Configure columns** - Allows changing the order and width of table columns and showing/hiding table columns. This action opens the following dialog box:

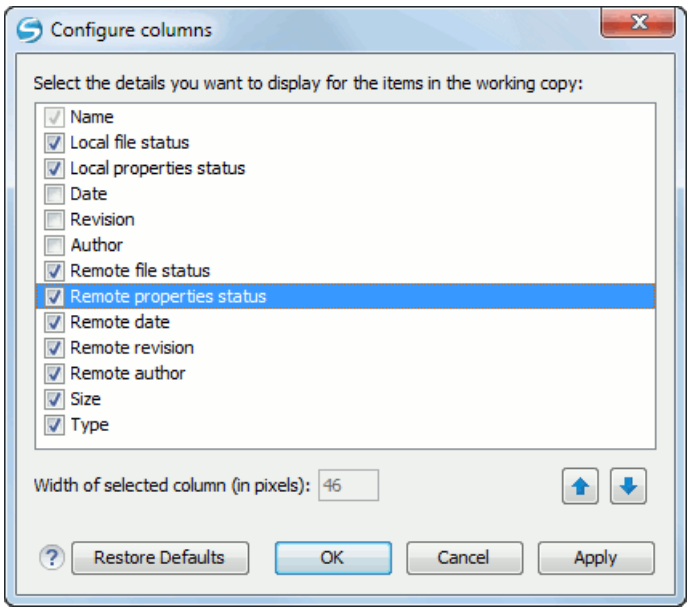


Figure 339: Configure Columns of Working Copy View

The order of the columns can be changed with the two arrow buttons. The column size can be edited in the **Width of selected column** field. The **Restore Defaults** button reverts all columns to the default order, width and enabled/disabled state from the installation of the application.

Working Copy Format

When a SVN working copy is loaded, Syncro SVN Client first checks the working copy's format. If it is a SVN 1.6 format, the admin data of that working copy is loaded and displayed in a tree like form in the view using the icons specific to the status of each resource: normal, unversioned, modified, etc. If it is the old format (SVN 1.5, SVN 1.4 or SVN 1.3), a confirmation dialog is displayed allowing the automatic conversion of the working copy to the new format, that is the SVN 1.6 one.

If you select the **Never ask me again** checkbox before pressing the **Yes** button, then the *Automatically upgrade working copies to the client's version* option is automatically checked. From now on, all other older version working copies will be automatically updated to the latest version.

The format of the working copy can be downgraded or upgraded at any time with the **Upgrade** and **Downgrade** actions available in the **Tools** menu. These actions allow switching between SVN 1.4, SVN 1.5 and SVN 1.6 working copy formats.

Refresh a Working Copy

A refresh is a frequent operation triggered automatically when you switch between two working copies using the toolbar selector of the **Working Copy** view and when you switch between Syncro SVN Client and other applications.

The **Working Copy** view features a fast refresh mechanism: the content is cached locally when loading the working copy for the first time. Later on, when the same working copy is displayed again, the application uses this cache to detect the changes between the cached content and the current content found on disk. The refresh operation is run on these changes only, thus improving the response time. improvement is noticeable especially when working with large working copies.

Contextual Menu Actions

The contextual menu in the Working Copy view contains the following actions:

- **Edit conflict (Ctrl + E)** - Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see *Edit conflicts*.
- **Open in Compare Editor (Ctrl + Alt + C)** - Displays changes made in the currently selected file.
- **Open (Ctrl + O)** - Opens the selected resource from the working copy. Files are opened with an internal editor or an external application associated with that file type, while folders are opened with a default file system browsing application (e.g. Windows Explorer on Windows, Finder on Mac OS X, etc).
- **Open with** submenu that allows you to open the selected resource either with <oXygen/> XML or with another application.
- **Expand all (Ctrl + Alt + X)** - Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.
- **Refresh (F5)** - Re-scans the selected resources recursively and refreshes their status in the working copy view.
- **Synchronize (Ctrl + Shift + S)** - Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.
- **Update (Ctrl + U)** - Updates the selected resources to the *HEAD* revision (latest modifications) from the repository. If the selection contains a directory, it will be updated depending on its *depth*.
- **Update to revision/depth** - Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the *sparse checkouts* section.

- **Commit** - Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.
- **Revert (Ctrl + Shift + V)** - Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Subversion's pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.
- **Override and Update** - Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.
- **Override and Commit** - Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.
- **Mark Resolved (Ctrl + Shift + R)** - Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.
- **Mark as Merged (Ctrl + Shift + M)** - Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.
- **Create patch (Ctrl + Alt + P)** - Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see *the section about patches*.
- **Compare with**:
  - **Latest from HEAD (Ctrl + Alt + H)** - Performs a 3-way diff operation between the selected file and the *HEAD* revision from the repository and displays the result in the **Compare view**. The common ancestor of the 3-way diff operation is the *BASE* version of the file from the local working copy.
  - **BASE revision (Ctrl + Alt + C)** - Compares the working copy file with the BASE revision file (the so-called *pristine copy*).
  - **Revision (Ctrl + Alt + R)** - Shows the **History view** containing the log history of that resource.
  - **Branch/Tag** - Compares the working copy file with a revision of the file from a branch or tag. The revision is specified by URL (selected with a repository browser dialog) and revision number (selected with a revision browser dialog).
  - **Each other** - Compares two selected files with each other.

  These *compare* actions are enabled only if the selected resource is a file.
- **Replace with**:
  - **Latest from HEAD** - Replaces the selected resources with their versions from the *HEAD* revision of the repository.
  - **BASE revision** - Replace the selected resources with their versions from the pristine copy (the BASE revision).

  Note: In some cases it is impossible to replace the current selected resources with their versions from the *BASE/HEAD* revision:
  - for **Replace with BASE revision** action, the resources being unversioned or added have no *BASE* revision, so they cannot be replaced. However, they will be deleted if the action is invoked on a parent folder. The action will never work for missing folders or for obstructing files (folders being obstructed by a file), because you cannot recover a tree of folders;
  - for **Replace with latest from HEAD** action, you must be aware that there are cases when resources will be completely deleted or reverted to BASE revision and after that updated to HEAD revision, in order to avoid conflicts. These cases are:
    - the resource is *unversioned*, *added*, *obstructed* or *modified*;
    - the resource is affected by a *svn:ignore* or *svn:externals* property which is locally added on the parent folder and not yet committed to the repository.

- **Show History (Ctrl + H)** - Displays the **History view** where the log history for the selected resource will be presented. For more details about resource history see the sections about *the resource history view* and *requesting the history for a resource*.
- **Show Annotation (Ctrl + Shift + A)** - It will display the **Annotations view** where all the users that modified the selected resource will be presented together with the specific lines and revision numbers modified by each user. For more details about resource annotations see *Annotations View*.
- **Revision Graph (Ctrl + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see *Revision Graph*.
- **Copy URL Location (Ctrl + Alt + U)** - Copies the encoded URL of the selected resource from the Working Copy to the clipboard.
- **Copy/Move to (Ctrl + M)** - Copies or moves to a specified location the currently selected resource.
- **Rename (F2)** - You can only rename one resource at a time. As for the move command, a copy of the original resource will be made with the new name and the original will be marked as deleted.
- **Delete (Delete)** - Allows you to delete resources from the Subversion working copy. If *unversioned*, *added* or *modified* resources will be encountered, a dialog will prompt you to confirm their deletion, because their content cannot be recovered. The action is not enabled when the selection contains *missing* resources.
- **New**:
  - **New File** - Creates a new file inside the selected folder. The newly created file will be added under version control only if the parent folder is already versioned.
  - **New Folder (Ctrl + Shift + F)** - Creates a child folder inside the selected folder. The newly created folder will be added under version control only if its parent is already versioned.
  - **New External Folder (Ctrl + Shift + W)** - Creates a new folder inside the selected folder, having the contents of a target folder from the current working copy's repository. The application does this by setting a *svn:externals* property on the selected folder and updating the folder in order to bring all the newly pointed resources inside your current working copy. The operation shows a dialog which allows you to specify the new folder's name and easily select the target folder by browsing the repository's contents.

    Subversion 1.5 and higher clients support relative external URLs. You can specify the repository URLs to which the external folders point using the following relative formats:

    - **../** - Relative to the URL of the directory on which the *svn:externals* property is set.
    - ^/ - Relative to the root of the repository in which the *svn:externals* property is versioned.
    - // - Relative to the scheme of the URL of the directory on which the *svn:externals* property is set.
    - / - Relative to the root URL of the server on which the *svn:externals* property is versioned.

- **Add to version control (Ctrl + Alt + V)** - Adds the selected resources to version control. A directory will be added recursively to version control. It is not mandatory to explicitly add resources to version control but it is recommended. At commit time unversioned resources will have to be manually selected in the commit dialog. This action is only active on unversioned resources.
- **Add to "svn:ignore" (Ctrl + Alt + I)** - Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the *svn:ignore* property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.
- **Cleanup (Ctrl + Shift + C)** - Performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.
- **Locking**:

- **Scan for locks (Ctrl + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see *Scanning for locks*.
- **Lock (Ctrl + K)** - Allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force(*steal*) the lock. The action is active only on files under version control. For more details on the use of this action see *Locking a file*.
- **Unlock (Ctrl + Alt + K)** - Releases(unlocks) the exclusive access to a file from the repository. You can also choose to unlock it by force(*break the lock*).

- **Show SVN Properties (Ctrl + P)** - Brings up the *Properties view* and displays the SVN properties for the selected resource.
- **File Information (Ctrl + I)** - Provides additional information for the selected resource from the working copy. For more details please see *Obtain information for a resource*.

Drag and Drop Operations

Files and folders can be added to the file tree of the Working Copy view as unversioned resources by drag and drop operations from other applications, for example Windows Explorer on Windows or Finder on Mac OS X. Also the structure of the files tree can be changed with drag and drop operations inside the view.

Assistant Actions

To ensure a continuous and productive work flow, when a view mode has no files to present, it offers a set of guiding actions with some possible paths to follow.

Initially, when there is no working copy configured the **All Files** view mode lists the following two actions:

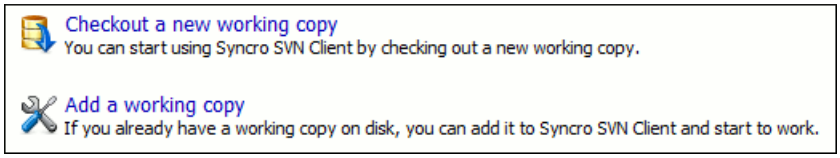


Figure 340: All Files Panel

For **Modified**, **Incoming**, **Outgoing**, **Conflicts** view modes, the following actions may be available, depending on the current working copy state in different contexts:

- Information message - Informs you why there are no resources presented in the currently selected view mode;
- **Synchronize with Repository** - Available only when there is nothing to present in the **Modified** and **Incoming** view modes;
- **Switch to Incoming** - Selects the **Incoming** view mode.
- **Switch to Outgoing** - Selects the **Outgoing** view mode.
- **Switch to Conflicts** - Selects the **Conflicts** view mode.
- **Show all changes/incoming/outgoing/conflicts** - Depending on the currently selected view mode, this action presents the corresponding resources after a synchronize operation was executed only on a part of the working copy resources.

Icons

The icons in the working copy view have a small overlaid icon which describes the current state of the resource in the working copy. These state icons are:

- **Modified** - The resource has been locally modified since the last update. This is obtained after editing a file and making changes.
- **External** - This indicates a mapping of a local directory to the URL of a versioned resource. It is declared with a *svn:externals* property in the parent folder.
- *Grayed* - A resource with a grayed icon but no overlaid icon is an ignored resource. It is obtained with the **Add to svn:ignore** action.
- **Switched** - This indicates a resource that has been switched from the initial repository location to a new location within the same repository. The resource goes to this state as a result of *the Switch action* executed from the contextual menu of the Working Copy view.
- **Immediate children** (*sparse checkout*) - The directory is marked with a purple bubble in the top left corner.
- **File children only** (*sparse checkout*) - The directory is marked with a blue bubble in the top left corner.
- **This folder only - empty** (*sparse checkout*)- The directory is marked with a gray bubble in the top left corner.

History View

In Subversion, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from any of the following three views: *Repository view menu* or *Working copy view menu*. From the **Working copy view** you can display the history of local versioned resources.

The view consists of three distinct areas:

- The revision table showing revision numbers, date/time of revision, the author's name, as well as the first line of the commit message;
- The list of resources affected by this revision (modified, added, deleted or changed properties);
- The commit message for the selected revision.

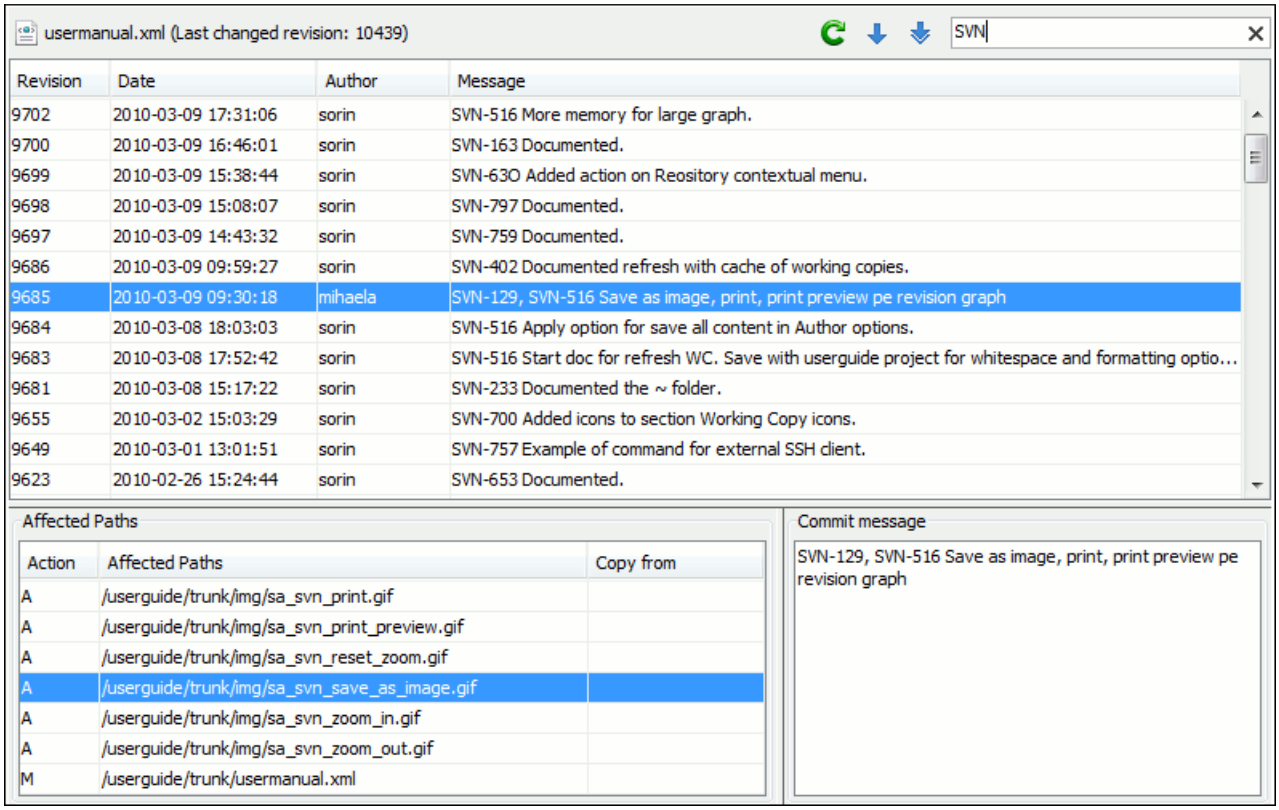


Figure 341: History View

The History Filter Dialog

The **History view** does not always show all the changes ever made to a resource because there may be thousands of changes and retrieving the entire list can take a long time. Normally you are interested in the more recent ones. That is why you can specify the criteria for the revisions displayed in the **History view** by selecting one of several options presented in the **History** dialog which is displayed when you invoke the **Show History** action.

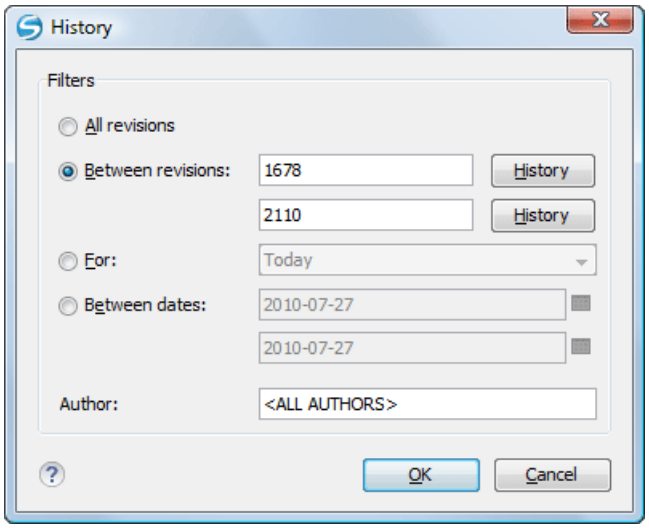


Figure 342: History Filters Dialog

Options for the set of revisions presented in the History view are:

- all revisions of the selected resource;

- only revisions between a start revision number and an end revision number;
- only revisions added in a period of time like today, last week, last month, etc.;
- only revisions between a start and an end date;
- only revisions committed by a specified SVN user.

The toolbar of the **History view** has two buttons for extending the set of revisions presented in the view: **Get next 50** and **Get all**.

The History Filter Field

When only the history entries which contain a specified substring need to be displayed in the **History view** the filter field displayed at the top of this view is the perfect fit. Just enter the search string in the field next to the label **Find**. Only the items with an author name, commit message, revision number or date which match the search string are kept in the **History view**. The filter action is executed and the content of the table is updated when the button  **Search** is pressed.

Features

Single selection actions:

- **Compare with working copy** - Compares the selected revision with your working copy file. It is enabled only when you select a file.
-  **Open** - opens the selected revision of the file into the Editor. This is enabled only for files.
- **Open with ...** - Displays the *Open with...* dialog to specify the editor in which the selected file will be opened.
- **Get Contents** - Replaces the current version from the working copy with the contents of the selected revision from the history of the file. The *BASE* version of the file is not changed in the working copy so that after this action the file will appear as modified in a synchronization operation, that is newer than the *BASE* version, even if the contents is from an older version from history.
- **Save revision to** - Saves the selected revision to a file so you have an older version of that file. This option is only available when you access the history of a file, and it saves a version of that file only.
- **Revert changes from this revision** - Reverts changes made in the selected revision.
- **Update to revision** - Updates your working copy resource to the selected revision.
- **Check out from revision** - Gets the content of the selected revision for the resource into local file system.
-  **Show Annotation** - Computes the latest revision number and author name that modified each line of the file up to the selected revision, that is no modification later than the selected revision is taken into account.
- **Change Author** - Changes the name of the SVN user that committed the selected revision.
- **Change Message** - Changes the commit message of the selected revision.

Double selection actions:

- **Compare revisions** - When the resource is a file, the action compares the two selected revisions using the **Compare view**. When the resource is a folder, the action displays the set of all resources from that folder that were changed between the two revision numbers.
- **Revert changes from these revisions** - Similar to the `svn-merge` command, it merges two selected revisions into the working copy resource. This action is only enabled when the resource history was requested for a working copy item.

For more information about the **History view** and its features please read the sections *Request history for a resource* and *Using the resource history view*

The Editor Panel of SVN Client

You can open a file for editing in an internal built-in editor. There are default associations between frequently used file types and the internal editors in *the File Types preferences panel*.

The internal editor can be accessed either from the *Working copy view* or from the *History view*. No actions that modify the content are allowed when the editor is opened with a revision from history.

Only one file at a time can be edited in an internal editor. If you try to open another file it will be opened in the same editor window. The editor provides syntax highlighting for known file types. This means that a different color will be used for each recognized token type found in the file. If the file's content type is unknown you will be prompted to choose the proper way the file should be opened.

After editing the content of the file in an internal editor you can save it to disk by using the **Save** action from the *File* menu or the Ctrl + S key shortcut. After saving your file you can see the file changed status in *the **Working Copy** view*.

If the internal editor associated with a file type is not the XML Editor, then the encoding set in *the preference Encoding for non XML files* is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the file's encoding.

Annotations View

Sometimes you need to know not only what was changed in a file, but also who made those changes. This view displays the author and the revision that changed every line in a file. Just click on a line in the editor panel where the file is opened to see the revision that edited that line last time highlighted in the **History view** and to see all the lines changed by that revision highlighted in the editor panel. Also the entries of the **Annotations view** corresponding to that revision are highlighted. So the **Annotations view**, the **History view** and the editor panel are synchronized. Clicking on a line in one of them highlights the corresponding lines in the other two.

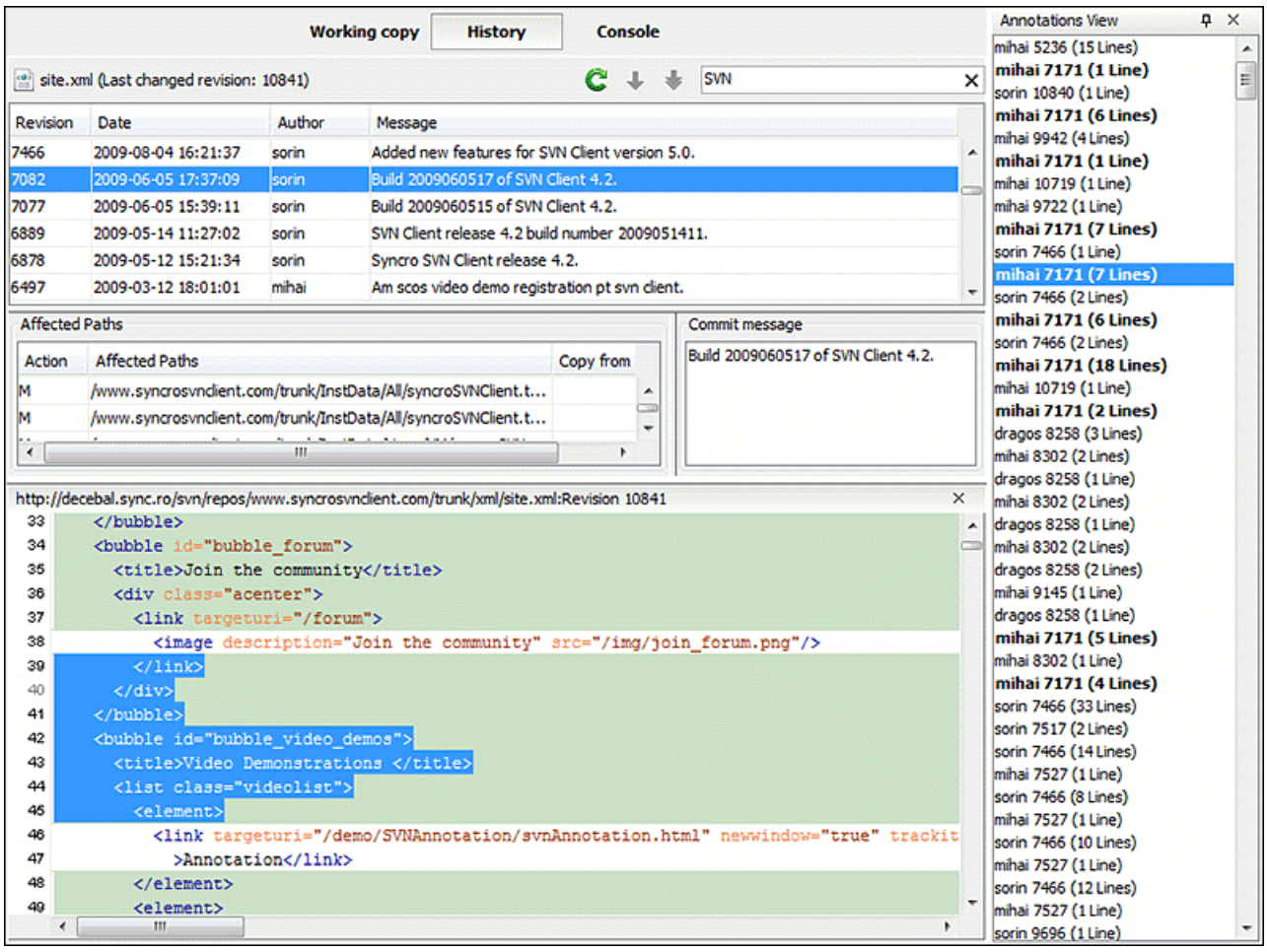


Figure 343: The Annotations View

The annotations of a file are computed with the **Show Annotation** action available on the right click menu of *the History view* and *the Repository view*.

If the file has a very long history, the computation of the annotation data can take long. If you want only the annotations of a range of revisions you can specify the start revision and the end revision of the range in a dialog similar with *the History filter dialog* that will be displayed in *the History view*. The action is called **Show Annotation** and is available on the right click menu of *the Working Copy view*.

Compare View

In the Syncro SVN Client there are three types of files that can be checked for differences: text files, image files and binary files. For the text files and image files you can use the built-in **Compare view**.

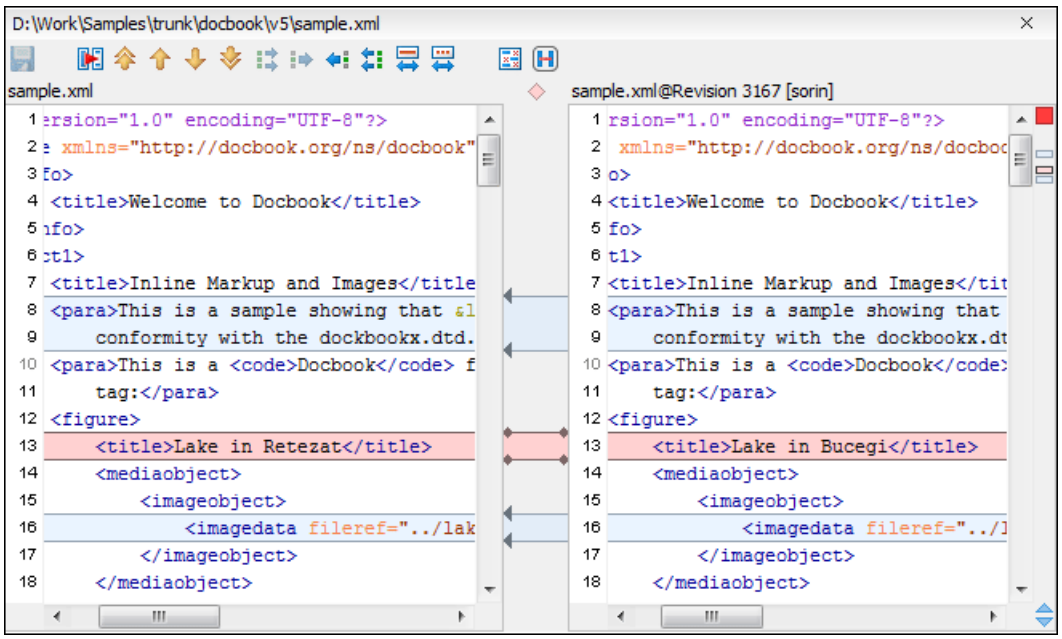


Figure 344: Compare View

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

When comparing text, the differences are computed using a *line differencing algorithm*. The view can be used to show the differences between two files in the following cases:

- after obtaining the outgoing status of a file with a **Refresh** operation, the view can be used to show the differences between your working file and the pristine copy. In this way you can find out what changes you will be committing;
- after obtaining the incoming and outgoing status of the file with the **Synchronize** operation, you can examine the exact differences between your local file and the *HEAD* revision file;
- you can use the **Compare view** from the **History view** to compare the local file and a selected revision or compare two revisions of the same file.

The Compare view contains two editors. Edits are allowed only in the left editor and only when it contains the working copy file. To learn more about how the view can be used in the day by day work see *View differences*.

Toolbar

The list of actions available in the toolbar consists of:

- **Save action** - Saves the content of the left editor when it can be edited.
- **Perform files differencing** - Performs files differencing on request.
- **Go to first modification** - Navigates to the first found difference.
- **Go to previous modification** - Navigates to the previous found difference.
- **Go to next modification** - Navigates to the next found difference.
- **Go to last modification** - Navigates to the last found difference.
- **Copy change from right to left** - Copies the selected change from the right editor to the left editor.
- **Copy all non-conflicting changes from right to left** - Copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.

- **Show modification details at word level** - Compares the currently selected change at word level to provide more details about that change.
- **Show modification details at character level** - Compares the currently selected change at character level to provide more details about that change.
- **Ignore whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.
- **Synchronized scrolling** - Scrolls both the views of the **Compare** view at the same time so that the two sides of a difference are visible at the same time.

These actions are available also from the *Compare* menu.

Image Preview

You can view your local files by using the built-in **Image preview** component. The view can be accessed from the *Working copy view* or from the *Repository view*. It can also be used from the *History view* to view a selected revision of a image file.

Only one image file can be opened at a time. If an image file is opened in the *Image preview* and you try to open another one it will be opened in the same window. Supported image types are *GIF*, *JPEG/JPG*, *PNG*, *BMP*. Once the image is displayed in the **Image preview** panel using the actions from the contextual menu one can scale the image at its original size (**1:1** action) or scale it down to fit in the view's available area (**Scale to fit** action).

Compare Images View

The images are compared using the Compare images view. The images are presented in the left and right part of the view, scaled to fit the view's available area. You can use the contextual menu actions to scale the images at their original size or scale them down to fit the view's available area.

The supported image types are: *GIF*, *JPG / JPEG*, *PNG*, *BMP*.

Properties View

The properties view presents Subversion properties for the currently selected resource from either the **Working Copy** view or the **Repositories** view.

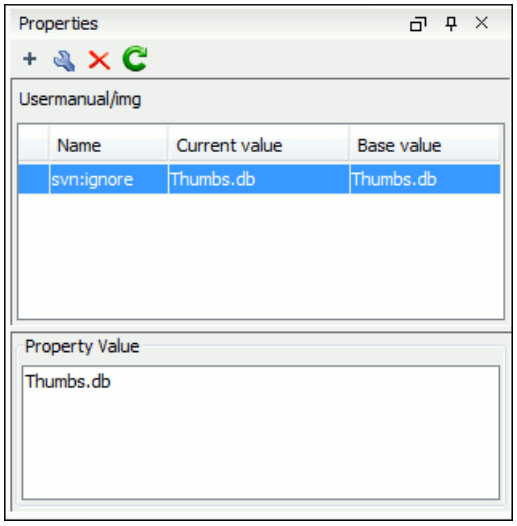


Figure 345: The Properties View

Above the table it is specified the currently active resource for which the properties are presented. Here you will also find a warning when an unversioned resource is selected.

The table in which the properties are presented has four columns:

- **State** - can be one of:
  - (empty) - normal unmodified property, same current and base values;
  - *(asterisk) - modified property, current and base values are different;
  - +(plus sign) - new property;
  - -(minus sign) - removed property.
- **Name** - the property name.
- **Current value** - the current value of the property.
- **Base value** - the base(original) value of the property.

The `svn:externals` Property

The `svn:externals` property can be set on a folder or a file. In the first case it stores *the URL of a folder from other repository*.

In the second case it stores the URL of a file from other repository. The external file will be added into the working copy as a versioned item. There are a few differences between directory and file externals:

- The path to the file external must be in a working copy that is already checked out. While directory externals can place the external directory at any depth and it will create any intermediate directories, file externals must be placed into a working copy that is already checked out.
- The external file URL must be in the same repository as the URL that the file external will be inserted into; inter-repository file externals are not supported.
- While commits do not descend into a directory external, a commit in a directory containing a file external will commit any modifications to the file external.

The differences between a normal versioned file and a file external:

- File externals cannot be moved or deleted; the `svn:externals` property must be modified instead; however, file externals can be copied.

A file external shows up as a X in the switched status column.

Attention:

Incomplete support - In subversion 1.6 it is not possible to remove a file external from your working copy once you have added it, even if you delete the `svn:externals` property altogether. You have to checkout a fresh working copy to remove the file.

Toolbar / Contextual Menu

The properties view toolbar and contextual menu contain the following actions:

- **Add a new property** - This button invokes the *Add property* dialog in which you can specify the property name and value.
- **Edit property** - This button invokes the *Edit property* dialog in which you can change the property value and also see its original(base) value.
- **Remove property** - This button will prompt a dialog to confirm the property deletion. You can also specify if you want to remove the property recursively.
- **Refresh** - This action will refresh the properties for the current resource.

Console View

The **Console View** shows the traces of all the actions performed by the application. Part of the displayed messages mirror the communication between the application and the Subversion server. The output is expressed as subcommands to the Subversion server and simulates the Subversion command-line notation. For a detailed description of the Subversion console output read the **SVN User Manual**.

The view has a simple layout, with most of its space occupied by a message area. On its right side, there is a toolbar holding the following buttons:

- **Clear** - Erases all the displayed messages;
- **Lock scroll** - Disables the automatic scrolling when new messages are appended in the view.

The maximum number of lines displayed in the console (length of the buffer) can be modified in the *Preferences* page. By default this value is set to 100.

Dynamic Help View

Dynamic Help view is a help window that changes its content to display the help section referring to the currently selected view. As you change the focused view, you are able to read a short description of it and its functionality.

The Revision Graph of a SVN Resource

The history of a SVN resource can be watched on a graphical representation of all the revisions of that resource together with the tags in which the resource was included. The graphical representation is identical to a tree structure and very easy to follow.

The graphical representation of a resource history is invoked with the **Revision graph** action available on the right click menu of a SVN resource in *the Working Copy view* and *the Repository view*.

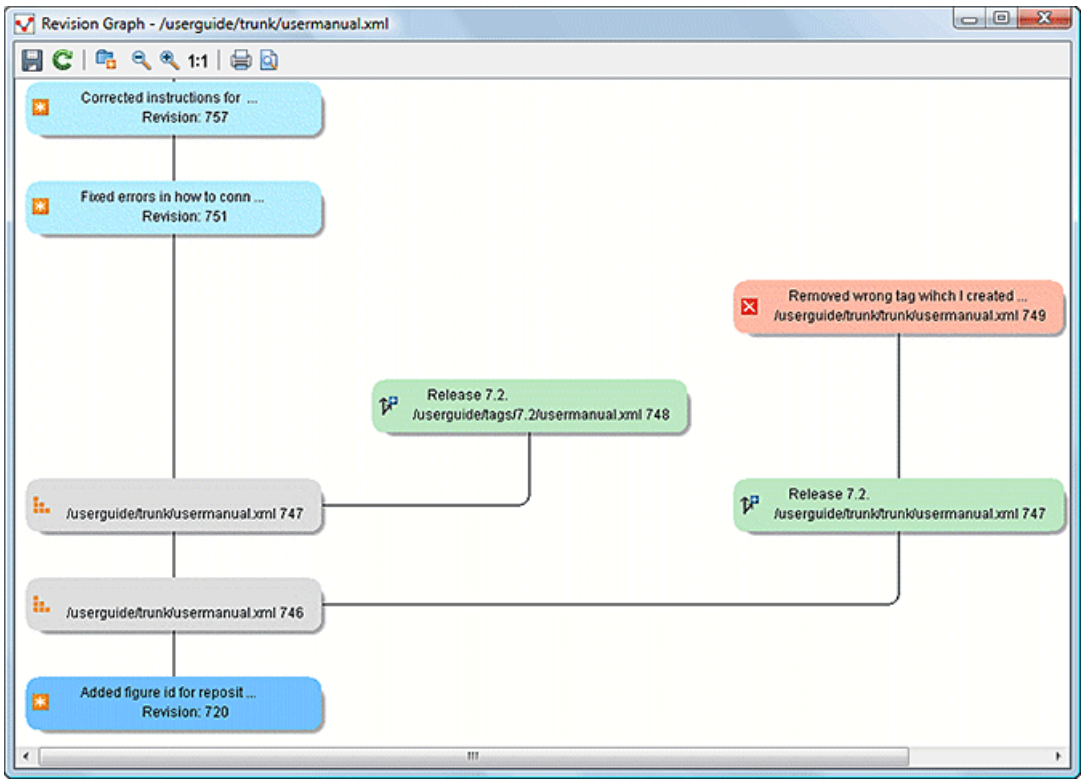


Figure 346: The Revision Graph of a File Resource

In every node of the revision graph an icon and the background color represent the type of operation that created the revision represented in that node. Also the commit message associated with that revision, the repository path and the revision number are contained in the node. The tooltip displayed when the mouse pointer hovers over a node specifies the URL of the resource, the SVN user who created the revision of that node, the revision number, the date of creation, the commit message, the modification type and *the affected paths*.

The types of nodes used in the graph are:

- **Added resource** - the icon for a new resource added to the repository ( ) and green background;
- **Copied resource** - the icon for a resource copied to other location, for example when a SVN tag is created ( ) and green background;
- **Modified resource** - the icon for a modified resource ( ) and blue background;
- **Deleted resource** - the icon for a resource deleted from the repository ( ) and red background;
- **Replaced resource** - the icon for a resource removed and replaced with another one on the repository ( ) and orange background;
- **Indirect resource** - the icon for a revision from where the resource was copied or an indirectly modified resource, that is a directory in which a resource was modified ( ) and grey background; the *Modification type* field of the tooltip specifies how that revision was obtained in the history of the resource.

A directory resource is represented with two types of graphs:

- **simplified graph** - lists only the changes applied directly to the directory;
- **complete graph** - lists also the indirect changes of the directory resource, that is the changes applied to the resources contained in the directory.

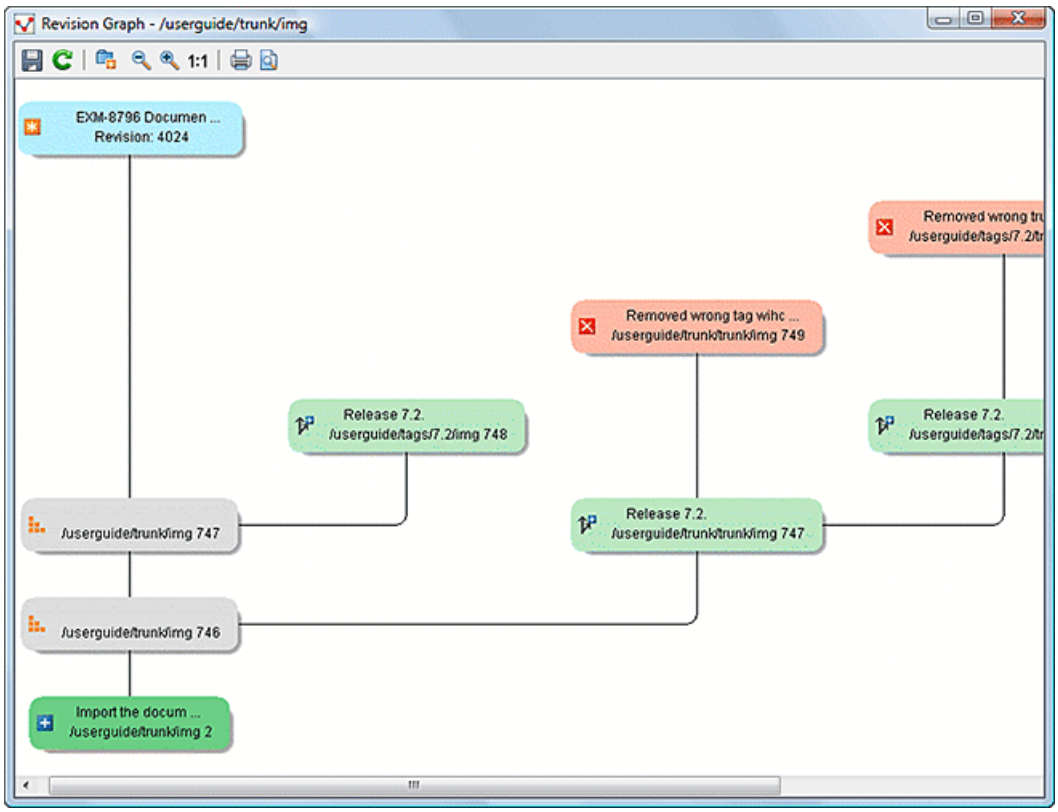


Figure 347: The Revision Graph of a Directory (Direct Changes)

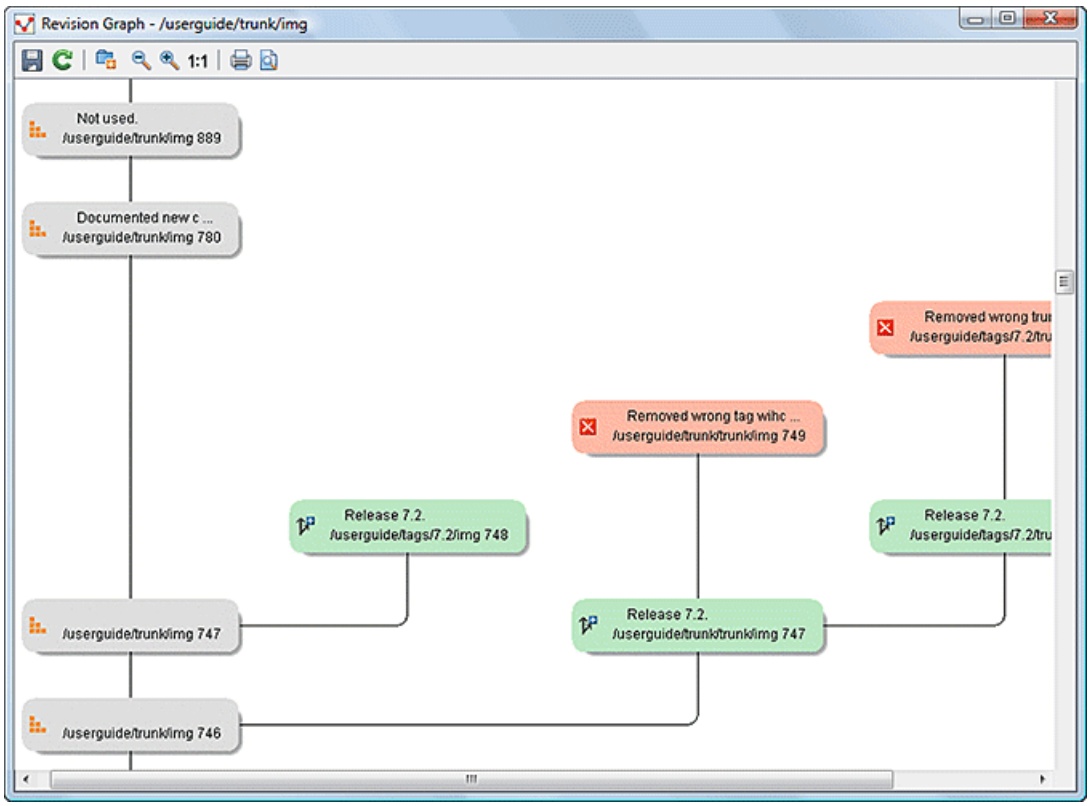


Figure 348: The Revision Graph of a Directory (Also Indirect Changes)

The **Revision graph** dialog toolbar contains the following actions:

- **Save as image** - Saves the graphical representation as image. For a large revision graph you have to *set more memory in the startup script*. The default memory size is not enough when there are more than 100 revisions that are included in the graph.
- **Show/Hide indirect modifications** - Switches between simplified and complete graph.
- **Zoom In** - Zooms in the graph.
- **Zoom Out** - Zooms out the graph. When the font reaches its minimum size, the graph nodes will display only the icons, leading to a very compact representation of the graph.
- **Reset scale** - Resets the graphical scale to a default setting.
- **Print** - Prints the graph.
- **Print preview** - Offers a preview of the graph to allow you to check the information to be printed.

Right clicking any of the graph nodes display a contextual menu containing the following actions:

- **Open** - Opens the selected revision in the editor panel. Available only for files.
- **Open with...** - Opens the selected revision in the editor panel. Available only for files.
- **Compare with HEAD** - Compares the selected revision with the HEAD revision and displays the result in the diff panel. Available only for files.
- **Show History** - Displays the history of the resource in *the History view*. Available for both files and directories.
- **Check Out** - *Checks out* the selected revision of the directory. Available only for directories.

When two nodes are selected in the revision graph of a file the right click menu of this selection contains only the **Compare** for comparing the two revisions corresponding to the selected nodes. If the resource for which the revision graph was built is a folder then the right click menu displayed for a two nodes selection also contains the **Compare** action but it computes the differences between the two selected revisions as a set of directory changes. The result is displayed in the *Directory Change Set* view in the same way as for *the compare action invoked from the History view* on two revisions of a folder.

Attention:

Generating the revision graph of a resource with many revisions may be a slow operation. You should enable caching for revision graph actions so that future actions on the same repository will not request the same data again from the SVN server which will finish the operation much faster.

Syncro SVN Client Preferences

The options used in the SVN client are saved and loaded independently from the Oxygen XML Editor options. However if at the Syncro SVN Client's first startup it cannot be determined a set of SVN options to be loaded, some of the preferences are imported from the XML Editor options (e.g. License key and HTTP Proxy settings).

The preferences dialog can be accessed from the *Options* -> Preferences. The preferences panels are called *Global* , *SVN* , *Diff colors* and *HTTP/Proxy Configuration.*

There is a second set of preferences applied to the SVN client: the preferences set in the global SVN files called 'config' and 'servers'. These are the files holding parameters that act as defaults applied to all the SVN client tools that are used by the same user on his/hers login account on the computer. These files can be opened for editing with the two edit actions available in the SVN client tool on the **Global Runtime Configuration** submenu of the **Options** menu.

Command Line Reference

This section specifies the equivalent Subversion commands for each action available in the graphical user interface of Syncro SVN Client.

Checkout Command

Used to pull a SVN tree from the server to the local file system. The syntax of checkout command is the following.

`svn checkout` --revision rev *URL* *PATH*

rev	The desired revision number (optional)	
URL		Repository URL you want to check out from.
PATH		Checkout target on file system.

Update Command

Brings changes from the repository into your working copy. The syntax of update command is the following.

`svn update` --revision rev *PATH*

rev	The desired revision number (optional)	
PATH		Checkout target on file system.

Updates resources to the last revision on which they were synchronized or to the *HEAD* revision, if no repository information is available.

Commit Command

Sends changes from your working copy to the repository.

`svn commit` -m "log message"--no-unlock *PATH*

-m "log message"	Specifies the commit comment.	
--no-unlock		Specifies that the resource should keep locks after commit if this is the case.
PATH		Location on the file system of the resource to commit. Can be more than one.

Diff Command

Displays the differences found between two revisions.

`svn diff` --revision rev1:rev2 *PATH*

rev1:rev2	Specifies the revisions to be compared.	
PATH		Location on the file system of the resource to be compared.

If you use the **Compare with latest from HEAD** from the *Working copy view* you will be comparing the local file with the HEAD revision file. If you use **Compare with BASE revision** the local file will be compared with the pristine copy. You can choose to compare the local file with an older revision or two revisions of the same file from the *History view*.

Show History

Display commit log messages.

`svn log` --revision rev1:rev2 --limit N --verbose *PATH*

rev1:rev2	Specifies the range of revisions for which to obtain the log.	
--limit N		Limits the number of messages brought to N.
--verbose		Gives detailed information about this command's execution.

Syncro SVN Client uses by default the *--limit* option in order to obtain only 50 log messages.

Refresh

Print the status of working copy files and directories.

`svn status` --verbose *PATH*

--verbose	Specifies that the status of all files should be reported.
PATH	Location on the file system of the resource to get status for.

Synchronize

`svn status` --show-updates *PATH*

--show-updates	Gets the resource status by contacting the repository.
PATH	Location on the file system of the resource to get status for.

Import

Commits an unversioned file or tree into the repository.

`svn import` -m "log message" *PATH* *URL*

-m "log message"	Specifies the commit log message.	
PATH		Local path to the resource on the file system.
URL	URL on the repository where the resource will be imported.	

Export

Exports a directory tree.

`svn export` --revision rev *URL* *PATH*

rev	Specifies the desired revision(if necessary).

URL		Repository URL you want to export from.
PATH		Location on the file system where to export.

Information

Displays information about a local or remote item.

`svn info` --revision rev *PATH*|*URL*

rev	Specifies the revision number for which the information will be requested.	
PATH		Local file system path to the resource.
URL		Repository URL for the resource.

Add

Add files, directories, or symbolic links.

`svn add` *PATH*...

PATH	Local file system path of the unversioned resources to be added to version control. More than one can be specified.

Add to svn:ignore

`svn propset` svn:ignore *PATH PARENTPATH*

svn:ignore	Predefined property name for ignoring resources.	
PATH		Relative path from the working copy root for the resource to be ignored.
PARENTPATH		Path to the parent of the resource to be ignored.

Delete

Deletes resources from a working copy or from a Subversion repository.

`svn delete` --recursive *PATH*|*URL*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Local file system path of the resource to delete.
PARENTPATH		Repository URL of the resource to delete.

Copy

Copy a file or directory in a working copy or in the repository.

`svn copy`*(SRCPATH DSTPATH)*|*(SRCURL DSTURL)*

SRCPATH	Working copy path of the resource to be copied.

DSTPATH		Working copy path where the resource will be copied to.
SRCURL	Repository path of the resource to be copied.	
DSTURL	Repository path where the resource will be copied to.	

Move / Rename

Move a file or directory.

`svn move`*(SRCPATH DSTPATH)*|*(SRCURL DSTURL)*

SRCPATH	Working copy path of the resource to be moved.	
DSTPATH		Working copy path where the resource will be moved to.
SRCURL	Repository path of the resource to be moved.	
DSTURL	Repository path where the resource will be moved to.	

Mark resolved

`svn resolved` --recursive *PATH*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Path to the resource in the local working copy.

Revert

Undo all local edits.

`svn revert` [--recursive] *PATH*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Local working copy path to revert to.

Cleanup

Recursively cleans up the working copy.

`svn cleanup` *PATH*

PATH	Local working copy path to clean up.

Show / Refresh Properties

`svn proplist` *PATH*

`svn propget` *PROPNAME PATH*

PATH	Local path of the resource.
PROPNAME	Property name.

First you can discover the property names with `svn proplist`, then you can obtain their values with `svn propget`.

Branch / Tag

`svn copy` -m "log message" *URL1* *URL2*

`svn copy` -m "log message" *URL1@rev1* *URL2*

`svn copy` -m "log message" *PATH* *URL*

-m "log message"	Commit message.	
URL1		Source repository URL.
rev1	Revision of the source.	
URL2	Destination repository URL.	
PATH	Source working copy path.	
URL	Destination repository URL.	

Merge

Apply the differences between two sources to a working copy path.

`svn merge` [--dry-run] *rev1:rev2* *URL* *PATH*

`svn merge` [--dry-run] *URL1@rev1* *URL2@rev2* *PATH*

--dry-run	Specifies that the operation will be simulated without making any modifications.	
URL		Repository URL for the resource to merge.
URL1	Repository URL for the start branch to merge.	
rev1	Start revision for the resource to merge.	
URL2	Repository URL for the end branch to merge.	
rev2	End revision for the resource to merge.	
PATH	Destination path in the working copy for the result of the merge.	

Scan for locks

Obtains the repository status for all the resources in the path.

`svn status` --show-updates --verbose *PATH*

--show-updates	Get the resource status by contacting the repository.	
--verbose		Specifies that the status of all files should be reported.
PATH		The location on the file system to get status for.

Lock

Lock working copy paths or URLs in the repository so that no other user can commit changes to them.

`svn lock` [--force] [-m "log message"] *PATH*

--force	Forces(steals) the lock.	
-m "log message"		Lock message.
PATH		Path to the file to be locked.

Unlock

Unlock working copy paths or URLs.

`svn unlock` [--force] *PATH*

--force	Forces(breaks) the lock.	
PATH		Path to the file from the working copy..

Mark as merged

`rename FILE` FILE.TMP

`svn update` *FILE*

`rename FILE.TMP FILE`

FILE	File to be marked as merged.	
FILE.TMP		Temporary filename.

Override and update

`svn revert` *PATH*

`svn update` *PATH*

PATH	Path of the resource to be overridden.

Override and Commit

If the resource is in conflict first you should perform a *Mark Resolved action*. If the resource has incoming changes you should perform a *Mark as Merged action* followed by a *Commit action*.

Add / Edit property

`svn propset` [--recursive] *PROPNAME PROPVALUE PATH*

--recursive	Specifies that the property should be set recursively.	
PROPNAME		Property name.
PROPVALUE	Property value.	

PATH		Resource's path.

Remove property

Removes a property from an item.

`svn propdel` [--recursive] *PROPNAME PATH*

--recursive	Specifies that the property should be deleted recursively.	
PROPNAME		Property name.
PATH		Resource's path.

Revert changes from this revision

`svn merge` rev:rev-1 *URL*

rev	Revision whose changes must be reverted.	
URL		The SVN URL corresponding to the resource.

Revert changes from these revisions

Short reference description.

`svn merge` rev1:rev2 *URL*

rev1	First revision number.	
rev2	Second revision number.	
URL		The SVN URL corresponding to the resource.

Chapter 22

Extending Oxygen with Plugins

Topics:

- *Introduction*
- *Oxygen Plugins*
- *How to Install a Plugin*
- *Example - A Selection Plugin*

This chapter explains how to write and install a plugin of the Oxygen XML Editor . It treats only the standalone version, as the Eclipse plugin version can be extended with other plugins following the rules of the Eclipse platform.

Introduction

Oxygen defines a couple of extension points to allow providing custom functionality via plugins. The plugin support includes the following types of plugins:

- General plugins
- Selection plugins
- Document plugins
- Custom protocol plugins
- Resource locking custom protocol plugins
- Components validation plugins
- Workspace access plugins
- Open redirect plugins

A selection plugin can be applied to both an XML document and a non-XML document. Other types of plugins can be applied only to XML documents.

A components validation plugin and a workspace access plugin are not connected with one document type, they have access to some resources of the application workspace used by all opened documents.

In order to develop a plugin a Java development environment must be installed. Apart from any library that the specific plugin requires, the file `oxygen.jar` is necessary for plugin compilation. Also an Oxygen installation is helpful for testing the deployment and plugin the functionality.

Oxygen Plugins

The Oxygen functionality can be extended with plugins that implement a clearly specified API. A plugin includes at least a descriptor file which is an XML file called `plugin.xml` and two Java classes that extend `ro.sync.exml.plugin.Plugin` and `ro.sync.exml.plugin.PluginExtension`. Most plugins work only in the Text mode of the XML editor panel while others work at the workspace level. For extending the Author mode of the XML editor panel, see *Author Developer Guide* for the custom Author actions API.

On the Oxygen website there is a *plugin development kit* with some sample plugins (source code and compiled code) and the Javadoc API necessary for developing custom plugins.

The minimal implementation of a plugin must provide:

- a Java class that extends the `ro.sync.exml.plugin.Plugin` class
- a Java class that implements the `ro.sync.exml.plugin.PluginExtension` interface
- a plugin descriptor file called `plugin.xml`

A `ro.sync.exml.plugin.PluginDescriptor` object is passed to the constructor of the subclass of the `ro.sync.exml.plugin.Plugin` class. It contains the following data items about the plugin:

- basedir - *File* object - the base directory of the plugin.
- description - *String* object - the description of the plugin.
- name - *String* object - the name of the plugin.
- vendor - *String* object - the vendor name of the plugin.
- version - *String* object - the plugin version.

The `ro.sync.exml.plugin.PluginDescriptor` fields are filled with information from the plugin descriptor file.

The plugin descriptor is an XML file that defines how the plugin is integrated in Oxygen and what libraries are loaded. The structure of the plugin descriptor file is fully described in a DTD grammar located in

`OXYGEN_INSTALLATION_FOLDER/plugins/plugin.dtd`. Here is a sample plugin descriptor used by the *Capitalize Lines* sample plugin:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
    name="Capitalize Lines"
    description="Capitalize the first character on each line"
    version="1.0.0"
    vendor="SyncRO"
    class="ro.sync.sample.plugin.caplines.CapLinesPlugin">
    <runtime>
        <library name="lib/caplines.jar"/>
    </runtime>
    <extension type="selectionProcessor"
    class="ro.sync.sample.plugin.caplines.CapLinesPluginExtension"
keyboardShortcut="ctrl shift EQUALS"/>
</plugin>
```

If your plugin is of type *Selection*, *Document* or *General*, and thus contributes an action either to the contextual menu or to the main menu, then you can assign a keyboard shortcut for it. You can use the `keyboardShortcut` attribute for each `extension` element to specify the desired shortcut.

Tip: To compose string representations of the desired shortcut keys you can go to the Oxygen **Menu Shortcut Keys** preferences page, press **Edit** on any action, press the desired key sequence and use the representation which appears in the edit dialog.

General Plugin

This plugin is the most general plugin type. It provides a limited API:

- The interface `GeneralPluginExtension` - Intended for general-purpose plugins, kind of external tools but triggered from the **Plugins** menu. The implementing classes must provide the method `process(GeneralPluginContext)` which must provide the plugin processing. This method takes as a parameter a `GeneralPluginContext` object.
- The class `GeneralPluginContext` - Represents the context in which the general plugin extension does its processing. The only method available is `getFrame()` which returns the currently editing frame (`java.awt.Frame`). It is useful if the plugin wants to display graphical components as they in general need a parent in order to appear properly on screen.

Selection Plugin

A selection plugin can be applied to both an XML document and a non-XML document. It works as follows: the user makes a selection in the editor, displays the contextual menu, and selects from the **Plugins** submenu the item corresponding to the plugin.

This plugin type provides the following API:

- The interface `SelectionPluginExtension` - The context containing the selected text is passed to the extension and the processed result is going to replace the initial selection. The `process(GeneralPluginContext)` method must return a `SelectionPluginResult` object which contains the result of the processing. The `String` value returned by the `SelectionPluginResult` object can include editor variables like *${caret}* and *${selection}*.
- The `SelectionPluginContext` object represents the context. It provides two methods:
  - `getSelection()` - Returns a `String` that is the current selection of text.
  - `getFrame()` - Returns a `Frame` that is the currently editing frame.

Document Plugin

This plugin type can be applied only to an XML document. It can modify the current document which is received as callback parameter.

The plugin is started by selecting the corresponding menu item from the contextual menu of the XML editor (Text mode), **Plugins** submenu. It provides the following API:

- The interface `DocumentPluginExtension` - Receives the context object containing the current document in order to be processed. The `process(GeneralPluginContext)` method can return a `DocumentPluginResult` object containing a new document.
- The `DocumentPluginContext` object represents the context. It provides two methods:
  - `getDocument()` - Returns a `javax.swing.text.Document` object that represents the current document.
  - `getFrame()` - Returns a `java.awt.Frame` object that represents the currently editing frame.

Custom Protocol Plugin

This type of plugins allows the developer to work with a custom designed protocol for retrieving and storing files.

It provides the following API:

- The interface `URLStreamHandlerPluginExtension` - There is one method that must be implemented:
  - `getURLStreamHandler(String protocol)` - It takes as an argument the name of the protocol and returns a `URLStreamHandler` object, or null if there is no URL handler for the specified protocol.
- With the help of the `URLChooserPluginExtension` interface, it is possible to write your own dialog that works with the custom protocol. This interface provides two methods:
  - `chooseURLs()` - Returns a `URL[]` object that contains the URLs the user decided to open with the custom protocol. You can invoke your own URL chooser dialog here and then return the chosen URLs having your own custom protocol.
  - `getMenuName()` - Returns a `String` object that is the name of the entry added in the **File** menu.
- With the help of the `URLChooserToolbarExtension` interface, it is possible to provide a toolbar entry which is used for launching the custom URLs chooser from the `URLChooserPluginExtension` implementation. This interface provides two methods:
  - `getToolbarIcon()` - Returns the `javax.swing.Icon` image used on the toolbar.
  - `getToolbarTooltip()` - Returns a `String` that is the tooltip used on the toolbar button.

Resource Locking Custom Protocol Plugin

This plugin type allows the developer to work with a custom designed protocol for retrieving and storing files. It can lock a resource on opening it in Oxygen. This type of plugin extends *the custom protocol plugin type* with resource locking support.

Such a plugin provides the following API:

- The interface `URLStreamHandlerWithLockPluginExtension` - The plugin receives callbacks following the simple protocol for resource locking and unlocking imposed by Oxygen.

  There are two additional methods that must be implemented:
  - `getLockHandler()` - Returns a `LockHandler` implementation class with the implementation of the lock specific methods from the plugin.
  - `isSupported(String protocol)` - Returns a `boolean` that is `true` if the plugin accepts to manage locking for a certain URL protocol scheme like `ftp`, `http`, `https`, or `customName`.

Components Validation Plugin

This plugin type allows the developer to make customization of the editor menus, toolbars, and some other components by allowing or filtering them from the user interface.

This plugin provides the following API:

- The interface `ComponentsValidatorPluginExtension` - There is one method that must be implemented:
  - `getComponentsValidator()` - Returns a `ro.sync.exml.ComponentsValidator` implementation class used for validating the menus, toolbars, and their actions.
- The interface `ComponentsValidator` provides methods to filter various features from being added to the application GUI:
  - `validateMenuOrTaggedAction(String[] menuOrActionPath)` - Checks if a menu or a tag action from a menu is allowed and returns a `boolean` value. A tag is used to uniquely identifying an action. The `String[]` argument is the tag of the menu / action and the tags of its parent menus if any.
  - `validateToolbarTaggedAction(String[] toolbarOrAction)` - Checks if an action from a toolbar is allowed and returns a *boolean* value. The `String[]` argument is the tag of the action from a toolbar and the tag of its parent toolbar if any.
  - `validateComponent(String key)` - Checks if the given component is allowed and returns a `boolean` value. The `String` argument is the tag identifying the component. You can remove toolbars entirely using this callback.
  - `validateAccelAction(String category, String tag)` - Checks if the given accelerator action is allowed to appear in the GUI and returns a `boolean` value. An accelerator action can be uniquely identified so it will be removed both from toolbars or menus. The first argument represents the action category, the second is the tag of the action.
  - `validateContentType(String contentType)` - Checks if the given content type is allowed and returns a `boolean` value. The `String` argument represents the content type. You can instruct the application to ignore content types like *text / xsl* or *text / xquery* and the application will no longer be able to recognize them.
  - `validateOptionPane(String optionPaneKey)` - Checks if the given options page can be added in the preferences option tree and returns a `boolean` value. The `String` argument is the option pane key.
  - `validateOption(String optionKey)` - Checks if the given option can be added in the option page and returns a `boolean` value. The `String` argument is the option key. This method is mostly used for internal use and it is not called for each option in a preferences page.
  - `validateLibrary(String library)` - Checks if the given library is allowed to appear listed in the **About** dialog and returns a `boolean` value. The `String` argument is the library. This method is mostly for internal use.
  - `validateNewEditorTemplate(EditorTemplate editorTemplate)` - Checks if the given template for a new editor is allowed and returns a `boolean` value. The `EditorTemplate` argument is the editor template. An `EditorTemplate` is used to create an editor for a given extension. You can thus filter what appears in the **New** dialog list.
  - `isDebuggerperspectiveAllowed()` - Check if the debugger perspective is allowed and returns a `boolean` value.
  - `validateSHMarker(String marker)` - Checks if the given marker is allowed and returns a `boolean` value. The `String` argument represents the syntax highlight marker to be checked. If you decide to filter certain content types, you can also filter the syntax highlight options so that the content type is no longer present in the Preferences options tree.

Tip: The best way to decide what to filter is to observe the values the application passes when these callbacks are called. You have to create an implementation for this interface which lists in the console all values received by each function. Then you can decide on the values to filter and act accordingly.

Workspace Access Plugin

Many complex integrations, like integrations with Content Management Systems (CMS) usually requires access to some workspace resources like the toolbar, menus and to the opened XML editors. This type of plugin is also useful because it allows you to make modifications to an opened editor's XML content.

The plugin must implement the interface `ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension`. The callback method `applicationStarted` of this interface allows access to a parameter of type `ro.sync.exml.workspace.api.standalone.StandalonePluginWorkspace` which in its turn allows for API access to the application workspace.

The interface `StandalonePluginWorkspace` has three methods which can be called in order to customize the toolbars, menus and views:

- `addToolbarComponentsCustomizer` - Contributes to or modifies existing toolbars. You can specify in the associated `plugin.xml` descriptor additional toolbar IDs using the following construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomWorkspaceAccess" ..............>
 <runtime>
  ........
 </runtime>

 <extension type="WorkspaceAccess" ...............>
 ................
  <toolbar id="SampleWorkspaceAccessToolbarID" initialSide="NORTH"
initialRow="1"/>
</plugin>
```

  The `toolbar` element adds a toolbar in the Oxygen interface and allows you to contribute your own plugin specific actions. The following attributes are available:

  - `id` - unique identifier of the plugin toolbar;
  - `initialSide` - specifies the place where the toolbar is initially displayed. The allowed values are NORTH and SOUTH.
  - `initialRow` - specifies the initial row on the specified side where the toolbar is displayed. For example the main menu has an initial row of "0" and the "Edit" toolbar has an initial row of "1".

  The `ro.sync.exml.workspace.api.standalone.ToolbarInfo` toolbar component information with the specified id will be provided to you by the customizer interface. You will thus be able to provide Swing components which will appear on the toolbar when the application starts.

- `addViewComponentCustomizer` - Contributes to or modifies existing views or contributes to the reserved custom view. You can specify in the associated `plugin.xml` descriptor additional view IDs using the following construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomWorkspaceAccess" ..............>
 <runtime>
  ........
 </runtime>

 <extension type="WorkspaceAccess" ...............>
 ................
 <view id="SampleWorkspaceAccessID" initialSide="WEST" initialRow="0"/>
</plugin>
```

  The `view` element adds a view in the Oxygen interface and allows you to contribute your own plugin specific UI components. The following attributes are available:

- `id` - unique identifier of the view component.
- `initialSide` - specifies the place where the view is initially displayed. The allowed values are NORTH, SOUTH, EAST and WEST.
- `initialRow` - specifies the initial row on the specified side where the view is displayed. For example the **Project** view has an initial row of `0` and the Outline view has an initial row of `1`. Both views are in the WEST part of the workbench.

The `ro.sync.exml.workspace.api.standalone.ViewInfo` view component information with the specified id will be provided to you by the customizer interface. You will thus be able to provide Swing components which will appear on the view when the application starts.

- `addMenuBarCustomizer` - Contributes to or modifies existing menu components.

Access to the opened editors can be done first by getting access to all URLs opened in the workspace using the API method `StandalonePluginWorkspace.getAllEditorLocations(int editingArea)`. There are two available editing areas: the DITA Maps Manager editing area where only DITA Maps are edited and the main editing area. Using the URL of an opened resource you can gain access to it using the `StandalonePluginWorkspace.getEditorAccess(URL location, int editingArea)` API method. A `ro.sync.exml.workspace.api.editor.WSEditor` allows then access to the current editing page. Special editing API is supported only for the **Text** (`ro.sync.exml.workspace.api.editor.page.text.WSTextEditorPage`) and the **Author** (`ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPage`) pages.

In order to be notified when editors are opened, selected and closed you can use the API method `StandalonePluginWorkspace.addEditorChangeListener` to add a listener.

Open Redirect Plugin

This type of plugin is useful for opening more than one file with only one open action. For example when a zip archive or an ODF file or an OOXML file is open in the **Archive Browser** view a plugin of this type can decide to open a file also from the archive in an XML editor panel. This file can be the `document.xml` main file from an OOXML file archive or a specific XML file from a zip archive.

The plugin must implement the interface `OpenRedirectExtension`. It has only one callback: `redirect(URL)` that receives the URL of the file opened by the Oxygen user. If the plugin decides to open also other files it must return an array of information objects (`OpenRedirectInformation[]`) that correspond to these files. Such an information object must contain the URL that is opened in a new editor panel and the content type, for example `text/xml`. The content type is used for determining the type of editor panel. A `null` content type allows auto-detection of the file type.

How to Write a CMS Integration Plugin

In order to have a complete integration between Oxygen and any CMS you usually have to write a plugin which combines two available plugin extensions:

- *Workspace Access*
- *Custom protocol*

The usual set of requirements for an integration between Oxygen and the CMS are the following:

- Contribute to the Oxygen toolbars and main menu with your custom **Check Out** and **Check In** actions:
  - **Check Out** triggers your custom dialogs which allow you to browse the remote CMS and choose the resources you want to open;
  - **Check In** allows you to send back to the server the modified content.

  You can use the **Workspace Access** plugin extension (and provided sample Java code) for all these operations.

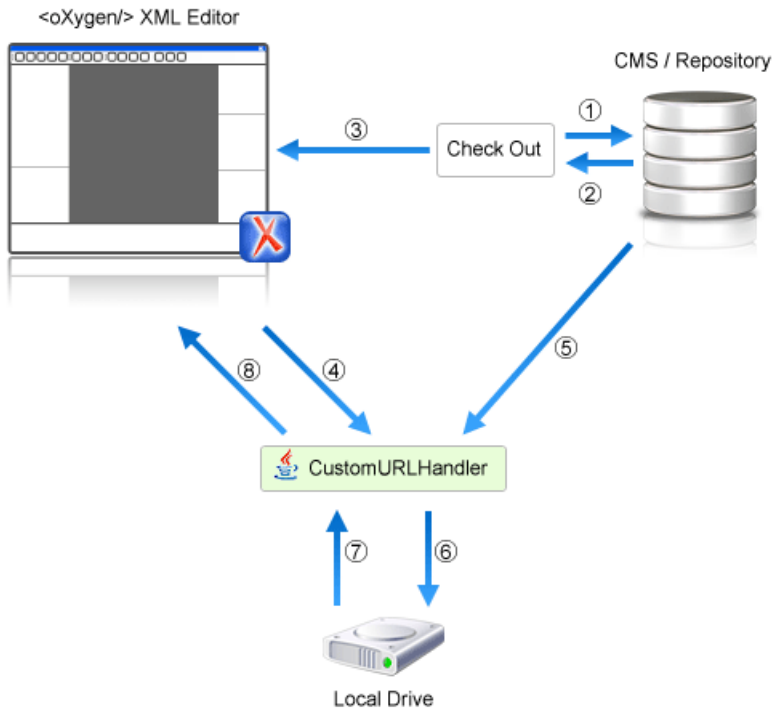
- When **Check Out** is called, use the Oxygen API to open your custom URLs (URLs created using your custom protocol). It is important to implement and use a **Custom Protocol** extension in order to be notified when the files are opened and saved and to be able to provide to Oxygen the content for the relative references the files may contain.

Your custom `java.net.URLStreamHandler` implementation checks out the resource content from the server, stores it locally and provides its content. Sample **Check Out** implementation:

```
  /**
   * Sample implementation for the "Check Out" method.
   *
   * @param pluginWorkspaceAccess The plugin workspace access (Workspace Access
 plugin).
   * @throws MalformedURLException
   */
 private void checkOut(StandalonePluginWorkspace pluginWorkspaceAccess) throws
 MalformedURLException {
    //TODO Show the user a custom dialog for browsing the CMS
    //TODO after the user selected the resource create an URL with a custom
protocol
    // which will uniquely map to the resource on the CMS using the URLHandler

    //something like:
    URL customURL = new URL("mycms://host/path/to/file.xml");
    //Ask Oxygen to open the URL
    pluginWorkspaceAccess.open(customURL);
    //Oxygen will then your custom protocol handler to provide the contents
for the resource "mycms://host/path/to/file.xml"
    //Your custom protocol handler will check out the file in a temporary
directory for example and provide the content from it.
    //Oxygen will also pass through your URLHandler if you have any relative
references which need to be opened/obtained.
  }
```

Here is a diagram of the **Check Out** process:



Each phase is described below:

1. Browse CMS repository
2. User chooses a resource
3. Use API to open custom URL: `mycms://path/to/file.xml`
4. Get content of URL: `mycms://path/to/file.xml`

5. Get content of resource
6. Store on disk for faster access
7. Retrieve content from disk if already checked out
8. Retrieved content

- Contribute a special **Browse CMS** action to every dialog in Oxygen where an URL can be chosen to perform a special action (like the **Insert a DITA Content Reference** action or the **Insert Image** action). Sample code:

```
      //Add an additional browse action to all dialogs/places where Oxygen
allows selecting an URL.
      pluginWorkspaceAccess.addInputURLChooserCustomizer(new
InputURLChooserCustomizer() {
       public void customizeBrowseActions(List<Action> existingBrowseActions,
 final InputURLChooser chooser) {
          //IMPORTANT, you also need to set a custom icon on the action for
situations when its text is not used for display.
          Action browseCMS = new AbstractAction("CMS") {
            public void actionPerformed(ActionEvent e) {
              URL chosenResource = browseCMSAndChooseResource();
              if (chosenResource != null) {
                try {
                 //Set the chosen resource in the dialog's combo box chooser.

                  chooser.urlChosen(chosenResource);
                } catch (MalformedURLException e1) {
                  //
                }
              }
            }
          };
          existingBrowseActions.add(browseCMS);
        }
      });
```

When inserting references to other resources using the actions already implemented in Oxygen, the reference to the resource is made by default relative to the absolute location of the edited XML file. You can gain control over the way in which the reference is made relative for a specific protocol like:

```
      //Add a custom relative reference resolver for your custom protocol.
     //Usually when inserting references from one URL to another Oxygen makes
 the inserted path relative.
      //If your custom protocol needs special relativization techniques then
it should set up a custom relative
      //references resolver to be notified when resolving needs to be done.
      pluginWorkspaceAccess.addRelativeReferencesResolver(
          //Your custom URL protocol for which you already have a custom
URLStreamHandlerPluginExtension set up.
          "mycms",
          //The relative references resolver
          new RelativeReferenceResolver() {
        public String makeRelative(URL baseURL, URL childURL) {
          //Return the referenced path as absolute for example.
          //return childURL.toString();
          //Or return null for the default behavior.
          return null;
        }
      });
```

- Write the `plugin.xml` descriptor. Your plugin combines the two extensions using a single set of libraries. The descriptor would look like:

```
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
```

```
 name="CustomCMSAccess"
 description="Test"
 version="1.0.0"
 vendor="ACME"
 class="custom.cms.CMSAccessPlugin">
 <runtime>
  <library name="lib/cmsaccess.jar"/>
 </runtime>
 <!--Access to add actions to the main menu and toolbars or to add custom
views.-->
 <!--See the
"ro.sync.sample.plugin.workspace.CustomWorkspaceAccessPluginExtension" Java
sample for more details-->
 <extension type="WorkspaceAccess"
  class="custom.cms.CustomWorkspaceAccessPluginExtension"/>
 <!--The custom URL handler which will communicate with the CMS
implementation-->
 <!--See the
"ro.sync.sample.plugin.workspace.customprotocol.CustomProtocolURLHandlerExtension"
 Java sample for more details-->
 <extension type="URLHandler"
  class="custom.cms.CustomProtocolURLHandlerExtension"/>
</plugin>
```

- Create a `cmsaccess.jar` JAR archive containing your implementation classes.
- Copy your new plugin directory in the `plugins` subfolder of the Oxygen install folder and start Oxygen.

Class Loading Issues

It is possible that the Java libraries you have specified in the plugin libraries list conflict with the ones already loaded by Oxygen. In order to instruct the plugin to prefer its libraries over the ones used by Oxygen, you can add the following attribute on the `<plugin>` root element:`classLoaderType="preferReferencedResources"` from the `plugin.xml` descriptor.

A **Late Delegation Class Loader** (the main class loader in Oxygen) is a `java.net.URLClassLoader` extension which prefers to search classes in its own libraries list and only if a class is not found there to delegate to the parent class loader.

The main Oxygen Class Loader uses as libraries all jars specified in the `OXYGEN_INSTALL_DIR\lib` directory. Its parent class loader is the default JVM Class loader. For each instantiated plugin a separate class loader is created having as parent the Oxygen Class Loader.

The plugin class loader can be either a standard `java.net.URLClassLoader` or a `LateDelegationClassLoader` (depending on the attribute `classLoaderType` in the `plugin.xml`). Its parent class loader is always the Oxygen `LateDelegationClassLoader`.

If you experience additional problems like the following:

```
java.lang.LinkageError: ClassCastException: attempting to cast
jar:file:/C:/jdk1.6.0_06/jre/lib/rt.jar!/javax/xml/ws/spi/Provider.classtojar:file:/D:/Program

    Files/Oxygen XML Editor

12/plugins/wspcaccess/../../xdocs/lib/jaxws/jaxws-api.jar!/javax/xml/ws/spi/Provider.class

    at javax.xml.ws.spi.Provider.provider(Provider.java:94) at
    javax.xml.ws.Service.<init>(Service.java:56)
.....................................................................
```

The cause could be the fact that some classes are instantiated using the context class loader of the current thread. The most straightforward fix is to write your code in a *try/finally* statement:

```
  ClassLoader oldClassLoader = Thread.currentThread().getContextClassLoader();
  try {
    //This is the implementation of the WorkspaceAccessPluginExtension plugin
interface.
    Thread.currentThread().setContextClassLoader(
        CustomWorkspaceAccessPluginExtension.this.getClass().getClassLoader());

    //WRITE YOUR CODE HERE
  } finally {
    Thread.currentThread().setContextClassLoader(oldClassLoader);
  }
```

How to Write A Custom Protocol Plugin

For creating a custom protocol plugin, apply the following steps:

1. Write the handler class for your protocol that implements the `java.net.URLStreamHandler` interface.

   Be careful to provide ways to correct and uncorrect the URLs of your files.
2. Write the plugin class by extending `ro.sync.exml.plugin.Plugin`.
3. Write the plugin extension class that implements the `ro.sync.exml.plugin.urlstreamhandler.URLStreamHandlerPluginExtension` interface.

   It is necessary that the plugin extension for the custom protocol implements the `URLStreamHandlerPluginExtension` interface. Without it, you cannot use your plugin, because Oxygen is not able to find the protocol handler.

   You can choose also to implement the `URLChooserPluginExtension` interface. It allows you to write and display your own customized dialog for selecting resources that are loaded with the custom protocol.

   An implementation of the extension `URLHandlerReadOnlyCheckerExtension` allows you to:

   - mark a resource as read-only when it is opened
   - switch between marking the resource as read-only and read-write while it is edited

   It is useful when opening and editing CMS resources.
4. Write the `plugin.xml` descriptor.

   Remember to set the name of the plugin class to the one from the second step and the plugin extension class name with the one you have chosen at step 3.
5. Create a .jar archive with all these files.
6. Install your new plugin in the `plugins` subfolder of the Oxygen install folder.

How to Install a Plugin

In the directory where Oxygen is installed there exists a directory called `plugins` that contains all the available plugins. In order for Oxygen to use the new functionality you provided, follow the next steps:

1. In the `plugins` folder create a subfolder to store the plugin files.
2. Put in this new folder the plugin descriptor file `plugin.xml`, the Java classes of the plugin and the other files that are referenced in the descriptor file.
3. Restart Oxygen.

Example - A Selection Plugin

The following plugin is called `UppercasePlugin` and is an example of *selection plugin*. It is used in Oxygen for capitalizing the characters in the current selection. This example consists of two Java classes and the plugin descriptor:

- `UppercasePlugin.java`:

```
package ro.sync.sample.plugin.uppercase;

import ro.sync.exml.plugin.Plugin;
import ro.sync.exml.plugin.PluginDescriptor;

public class UppercasePlugin extends Plugin {
    /**
    * Plugin instance.
    */
    private static UppercasePlugin instance = null;

    /**
    * UppercasePlugin constructor.
    *
    * @param descriptor Plugin descriptor object.
    */
    public UppercasePlugin(PluginDescriptor descriptor) {
        super(descriptor);

        if (instance != null) {
            throw new IllegalStateException("Already instantiated !");
        }
        instance = this;
    }

    /**
    * Get the plugin instance.
    *
    * @return the shared plugin instance.
    */
    public static UppercasePlugin getInstance() {
        return instance;
    }
}
```

- `UppercasePluginExtension.java`:

```
package ro.sync.sample.plugin.uppercase;

import ro.sync.exml.plugin.selection.SelectionPluginContext;
import ro.sync.exml.plugin.selection.SelectionPluginExtension;
import ro.sync.exml.plugin.selection.SelectionPluginResult;
import ro.sync.exml.plugin.selection.SelectionPluginResultImpl;

public class UppercasePluginExtension implements SelectionPluginExtension {
    /**
    * Convert the text to uppercase.
    *
    *@param  context  Selection context.
    *@return          Uppercase plugin result.
    */
    public SelectionPluginResult process(SelectionPluginContext context) {
        return new SelectionPluginResultImpl(
```

```
            context.getSelection().toUpperCase());
    }
}
```

- `plugin.xml`:

```
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
    name="UpperCase"
    description="Convert the selection to uppercase"
    version="1.0.0"
    vendor="SyncRO"
    class="ro.sync.sample.plugin.uppercase.UppercasePlugin">
    <runtime>
        <library name="lib/uppercase.jar"/>
    </runtime>
    <extension type="selectionProcessor"
     class="ro.sync.sample.plugin.uppercase.UppercasePluginExtension"/>
</plugin>
```

Chapter 23

Text Editor Specific Actions

Topics:

- *Undoing and Redoing User Actions*
- *Copying and Pasting Text*
- *Finding and Replacing Text in the Current File*
- *Finding and Replacing Text in Multiple Files*
- *Spell Checking*
- *Changing the Font Size*
- *Word/Line Editor Actions*
- *Dragging and Dropping the Selected Text*
- *Inserting a File at Caret Position*
- *Opening the File at Caret in System Application*
- *Opening the File at Caret Position*
- *Switching Between Opened Tabs*
- *Printing a File*
- *Exiting the Application*

The Text mode of the editor panel 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.

Undoing and Redoing User Actions

- **Undo** - menu **Edit** > **Undo (Ctrl+Z)** or the toolbar button **Undo** - Reverses a maximum of 100 editing actions to return to the preceding state. Complex operations like **Replace All**, **Indent selection**, etc are treated as a single undo event.
- **Redo** - menu **Edit** > **Redo (Ctrl+Y for Windows, Ctrl+Shift+Z for Mac OSX and Linux)** or the toolbar button **Redo** - Recreates a maximum of 100 editing actions that were undone by the **Undo** function.

Copying and Pasting Text

- **Edit** > **Cut (Ctrl+X)** or the toolbar button **Cut** - Removes the current selected node from the document and places it in the clipboard.
- **Edit** > **Copy (Ctrl+C)** or the toolbar button **Copy** - Places a copy of the current selection in the clipboard as RTF. All text attributes such as color, font or syntax highlight are preserved when pasting into another application.
- **Edit** > **Paste (Ctrl+V)** or the toolbar button **Paste** - Places the current clipboard content into the document at the cursor position.
- **Edit** > **Select All (Ctrl+A)** - Selects the entire body of the current document, including whitespace preceding the first and following the last character.

Finding and Replacing Text in the Current File

This section explains how to use the find and replace features of the application.

The Find / Replace Dialog

The **Find / Replace** dialog is opened from menu **Find** > **Find / Replace... (Ctrl+F)** or the toolbar button **Find / Replace**. It enables you to define search and replace operations on the current document. The find works on multiple lines, which means a find match can cover characters on more than one line. Special characters like newline and tab can be inserted using the contextual menu.

To insert a new line in the find or replace text area, press **(CTRL + Enter)** instead of **(Enter)**. The replace operation can bind Perl 5 regular expression group variables ($1, $2, etc.) from the find match. For example to replace the tag with attributes called `tag-name` with the tag `tag-name1` use *<tag-name(\s+)(.*)>* in the **Text to find** area and *<tag-name1$1$2>* in the **Replace with** area.

The following actions can be executed in the **Find / Replace** dialog:

- Find occurrences of a word or string of characters including white spaces, represented on one or multiple lines. Highlight their position in the editor.
- Replace occurrences of target defined in the **Text to find** area with a word or string of characters, including white spaces, that can be on a line or on multiple lines, defined in the **Replace with** area.
- Find all occurrences of a word or string of characters including white spaces that can be on a line or on multiple lines and display a result list to the message panel.
- Replace all occurrences of a word or string of characters including white spaces that can be on a line or on multiple lines.

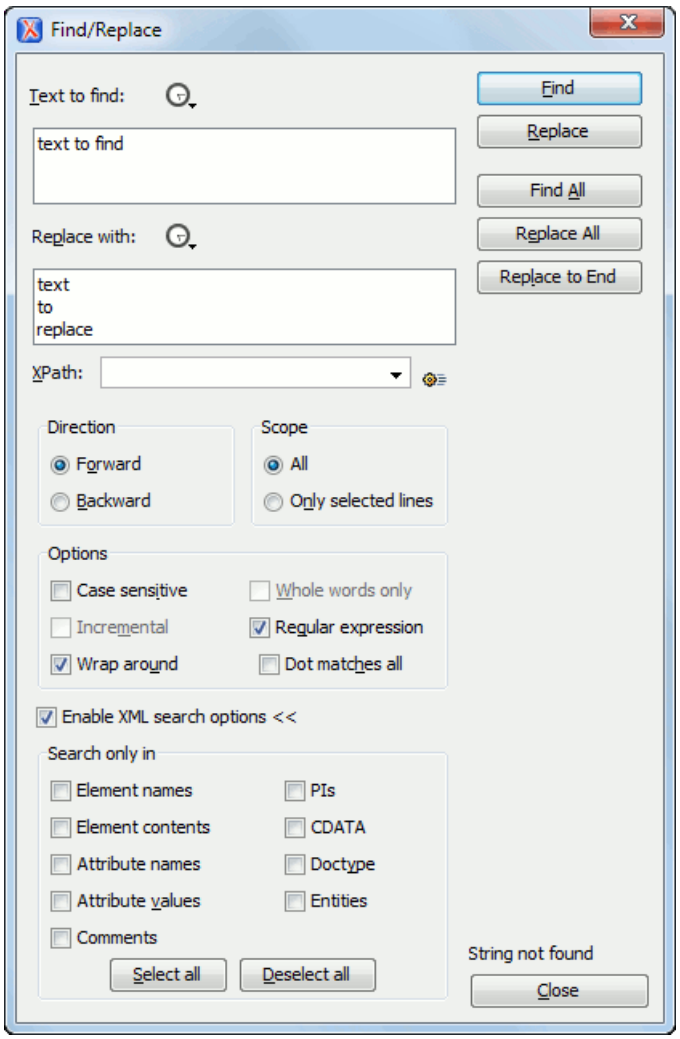


Figure 349: Find / Replace Dialog

The dialog contains the following options:

- **Text to find** - The target character string to search for. You can search for Unicode characters specified in the `\uNNNN` format. Also, hexadecimal notation (`\xNNNN`) and octal notation (`\0NNNN`) can be used. In this case you have to check the **Regular expression** checkbox. For example, to search for a space character you can use the `\u0020` code.
- **Replace with** - The character string with which to replace the target. The string for replace can be on a line or on multiple lines. Special characters like newline and tab can be inserted using the contextual menu. It may contain Perl 5 regular expression group markers, only if the search expression is a regular expression and the **Regular expression** checkbox is checked.

  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.

  Unicode characters can also be used in the **Replace with** area.
- The history buttons - The last find and replace operations history is available using the history buttons from the top of the find and replace text areas.
- **XPath** - The XPath 2.0 expression you input in this combo is used to restrict the search scope.

  The *content completion assistant* helps you input XPath expressions, valid in the current context.

- **Direction** - Specifies if the search direction is from current position to end of file (**Forward** direction) or to start of file (**Backward** direction).
- **Scope** - Specifies if the search is executed on all file or only on the lines that were selected when the dialog was invoked. If the selection was on a single line the search is executed on the whole file (by default the **All** option is selected).
- **Find** - Executes a find operation for the next occurrence of the target. It stops after highlighting the find match in the editor panel.
- **Replace** - Executes a replace operation for the target followed by a find operation for the next occurrence.
- **Find All** - Executes a find operation and displays all results to the message panel.
- **Replace All** - Executes a replace operation in the entire scope of the document.
- **Replace to End** - Executes a replace operation starting from current target until the end of the document, in the direction specified by the current selection of the **Direction** switch (**Forward** or **Backward**).
- **Case sensitive** - When checked, operations are case-sensitive.
- **Whole words only** - When checked, only whole occurrences of a word will be included in the operation.
- **Incremental** - When checked, the search operation is started every time you type or delete a letter in the **Text to find** text box.
- **Regular expression** - When checked, it allows you to use regular expressions in Perl 5 syntax. A content completion assistant window is available to help you edit regular expressions. It is activated every time you type \(backslash key) or on-demand if you press **Ctrl-Space**.
- **Dot matches all** - A dot used in a regular expression matches also end of line characters.
- **Wrap around** - Continues the find from the start (end) of the document after reaching the end (start) when the search is in forward (backward) direction.
- **Enable XML search options** - When checked, the dialog is enlarged and the XML search options are shown below. This option restricts the search domain to the checked XML component types.

  The supported XML component types are the following:

  - **Element names** - Only the names are searched, without '<', '/', '>', attributes or white-spaces. e. g.: <element />.
  - **Element contents** - Search in the text content of XML elements.
  - **Attribute names** - Only the attribute names are searched, without the leading or trailing white-spaces.
  - **Attribute values** - Only the attribute values are searched, without single quotes(') or double quotes("). e. g.: 'value' or "value"
  - **Comments** - Only the content of comments are searched, skipping '<!--', '-->'. e. g.: <!--Comment content-->
  - Processing Instructions (**PIs**) - Only the content are searched, skipping '<?', '?>'. e. g.: <?processing instruction?>
  - **CDATA** - Only the content are searched, skipping '<![CDATA[', ']]>'. e. g.: `<![CDATA[cdata content]]>`
  - **DOCTYPE** - Only the content of the DOCTYPE section are searched.
  - **Entities** - Only the entity names are searched.

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

  Note: Please note that since searching in some XML component types is performed only on their content skipping some of their headers / footers (see the list above), even if all the XML component types are checked, some filtering is still performed. To completely disable it you have to uncheck the option **Enable XML search options**.

- **Find All Elements / Attributes ...** - In Author mode an attribute cannot be searched directly. For finding an attribute just click on the link **Find All Elements / Attributes ...** in the dialog which opens the dialog with the same name.

The Find All Elements / Attributes Dialog

This dialog is opened from menu **Find** > **Find All Elements... (Ctrl+Shift+E)** or from the shortcut **Find All Elements / Attributes** that is available in *the **Find / Replace** dialog* . It assists you in defining XML elements / attributes search operations on the current document.

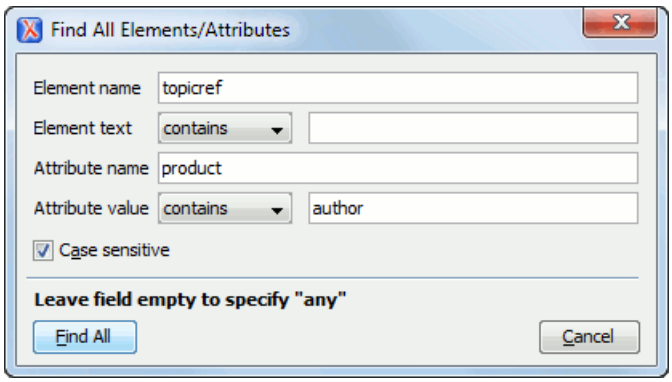


Figure 350: Find All Elements / Attributes dialog

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.

All these search criteria can be combined 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 dialog fields are described as follows:

- **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.

The Quick Find Toolbar

A reduced version of *the Find / Replace dialog* is available as a toolbar that is activated by the shortcut specified in the **Find** menu: **(Ctrl+Alt+F)** on Windows and Linux, **(Cmd+Alt+F)** on Mac OS X. The toolbar is displayed by default at the bottom of the Oxygen window, above the status bar.

The **Next**, **Previous**, **All**, **Incremental** and **Case sensitive** controls work *in the same way as in the Find / Replace dialog*. The search process works as if the **Search also in tags** option of *the Find / Replace dialog* is true, the **Whole words only** one is false, the **Regular expression** one is false and the **Wrap around** one is true. You can also use the last two toolbar actions to quickly open the *Find / Replace* and the *Find / Replace in Files* dialogs. The toolbar becomes invisible again when the **(ESC)** key is pressed.

The enabling shortcut can be changed in **Options** > **Preferences** > **Menu Shortcut Keys** > **Quick Find** . *As with any dockable toolbar,* the screen location of the Quick Find toolbar can be changed at any time by dragging (and docking) it to the desired location. However the buttons of this toolbar can be used only if it has a horizontal layout so docking it to the West side or the East side of the window is not allowed.

Keyboard Shortcuts for Finding the Next and Previous Match

Navigation from a find match to the next one or the previous one is very easy with two keyboard shortcuts: F3 and Shift F3. They are useful to quickly repeat the last find action performed with *the Find / Replace dialog,* taking into account the same find options set there through check boxes.

- **Find** > **Find Next (F3)** - Performs another search in forward direction using the last search configuration.
- **Find** > **Find Previous (Shift+F3)** - Performs another search in backward direction using the last search configuration.

Finding and Replacing Text in Multiple Files

The **Find / Replace in Files** dialog is opened from menu **Find** > **Find / Replace in Files...** or from the toolbar button  **Find / Replace in Files**. It enables you to define "search for" or "search for and replace" operations across a number of files. The find works at line level, which means a find match cannot cover characters on more than one line. The replace operation can bind Perl 5 regular expression group variables ($1, $2, etc.) from the find match. For example to replace the tag with attributes called `tag-name` with the tag `tag-name1` use as **Text to find** the expression *<tag-name(\s+)(.*)>* and as the **Replace with** expression *<tag-name1$1$2>*.

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

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

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

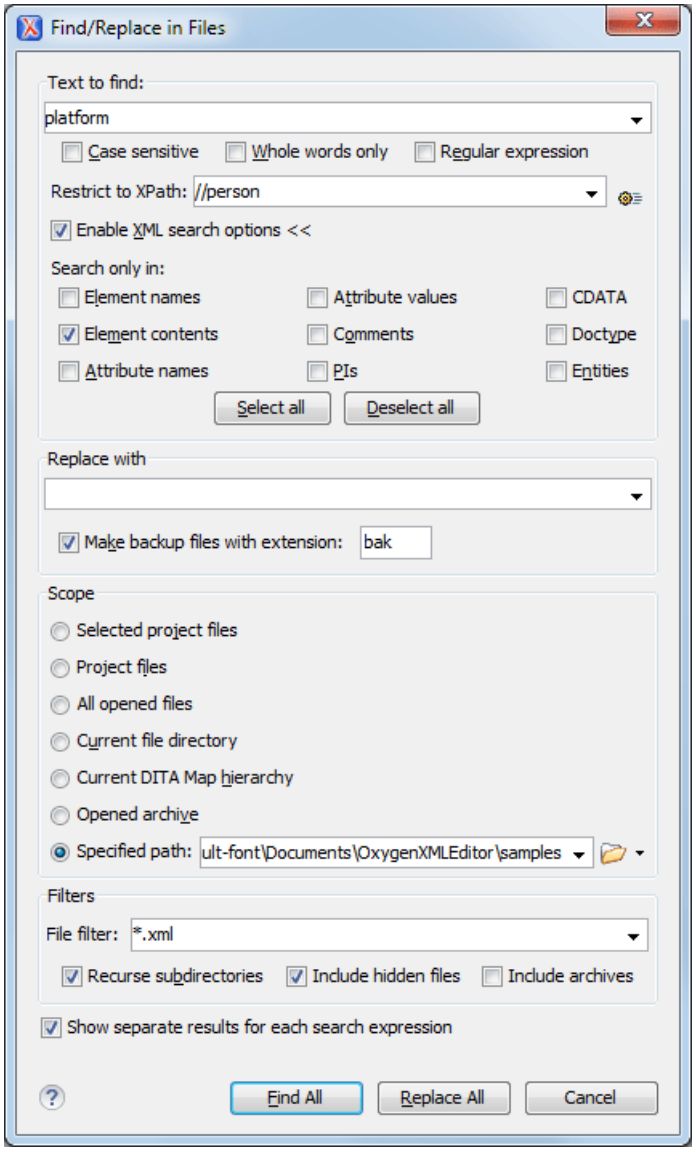


Figure 351: Find / Replace in Files

The dialog contains the following options:

- **Text to find** - The target character string to search for. You can search for Unicode characters specified in the `\uNNNN` format. Also, hexadecimal notation (`\xNNNN`) and octal notation (`\0NNNN`) can be used. In this case you have to check the **Regular expression** checkbox. For example, to search for a space character you can use the `\u0020` code.
- **Case sensitive** - When checked, operations are case-sensitive.
- **Whole words only** - When checked, only whole occurrences of a word will be included in the operation.
- **Regular expression** - When checked, it allows you to use regular expressions in Perl 5 syntax. A content completion assistant window is available to help you edit regular expressions. It is activated every time you type \(backslash key) or on-demand if you press **Ctrl-Space**.
- **XPath** - The XPath 2.0 expression you input in this combo is used to restrict the search scope.

  The *content completion assistant* helps you input XPath expressions, valid in the current context.
- **Enable XML search options** - When checked, the dialog is enlarged and the XML search options are shown below. This option restricts the search domain to the checked XML component types.

  The supported XML component types are the following:

- **Element names** - Only the names are searched, without '<', '/', '>', attributes or white-spaces. e. g.: <element />.
- **Element contents** - Search in the text content of XML elements.
- **Attribute names** - Only the attribute names are searched, without the leading or trailing white-spaces.
- **Attribute values** - Only the attribute values are searched, without single quotes(') or double quotes("). e. g.: 'value' or "value"
- **Comments** - Only the content of comments are searched, skipping '<!--', '-->'. e. g.: <!--Comment content-->
- Processing Instructions (**PIs**) - Only the content are searched, skipping '<?', '?>'. e. g.: <?processing instruction?>
- **CDATA** - Only the content are searched, skipping '<![CDATA[', ']]>'. e. g.: `<![CDATA[cdata content]]>`
- **DOCTYPE** - Only the content of the DOCTYPE section are searched.
- **Entities** - Only the entity names are searched.

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

Note: Please note that since searching in some XML component types is performed only on their content skipping some of their headers / footers (see the list above), even if all the XML component types are checked, some filtering is still performed. To completely disable it you have to uncheck the option **Enable XML search options**.

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

Note: The search is performed only on local files. If you have added to the project remote files from an FTP or WebDAV server, these files are skipped.

- **Project files** - Searches in all files from the current project. Not displayed when dialog is started from contextual menu of ***DITA Maps*** *view* and *Archive Browser* view.
- **All opened files** - Searches in all files opened in Oxygen (regular files or DITA Maps). You are prompted to save all modified files before any operation is performed. Not displayed when dialog is started from contextual menu of ***DITA Maps*** *view* and *Archive Browser* view.
- **Current file directory** - The search is done in the directory of the file opened in the current editor panel. If there is no opened file, this option is disabled in the dialog. Not displayed when dialog is started from contextual menu of ***DITA Maps*** *view* and *Archive Browser* view.
- **Current DITA Map hierarchy** - The search is done in all maps and topics referenced by the current edited DITA Map. If **Recurse referenced maps** option is checked, the references from the maps referenced in the main DITA map are also searched in. If there is no opened DITA Map, this option is disabled in the dialog. You are prompted to save all modified files before any operation is performed. Displayed only when dialog is opened from ***DITA Maps*** *view*.
- **Opened archive** - The search is done in archive opened in *Archive Browser* view. Displayed only when dialog is opened from *Archive Browser* view.
- **Specified path** - Chooses the search path.
- **File filter** - Narrows the scope of the operation only to the files that match the given filters.
- **Recurse subdirectories** - When checked, the search is performed recursively in the sub directories found in the specified directory path.
- **Include hidden files** - When checked, the search is performed also in the hidden files.
- **Include archives** - When checked, the search is also done in all individual file entries from all supported ZIP-type archives.
- **Show separate results for each search expression** - When checked, the application opens a new tab to display the result of each new search expression. When the option is unchecked, the search results are displayed in the *Find in Files* tab, replacing any previous search results.
- **Find All** - Executes a find operation and returns the result list to the message panel.

- **Replace All** - Replaces all occurrences of the target contained in the specified files.

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

Spell Checking

The **Spelling** dialog enables you to check the spelling of the current document. It is opened from menu **Edit** > **Check Spelling (F4) (** or the toolbar button **Check Spelling**.

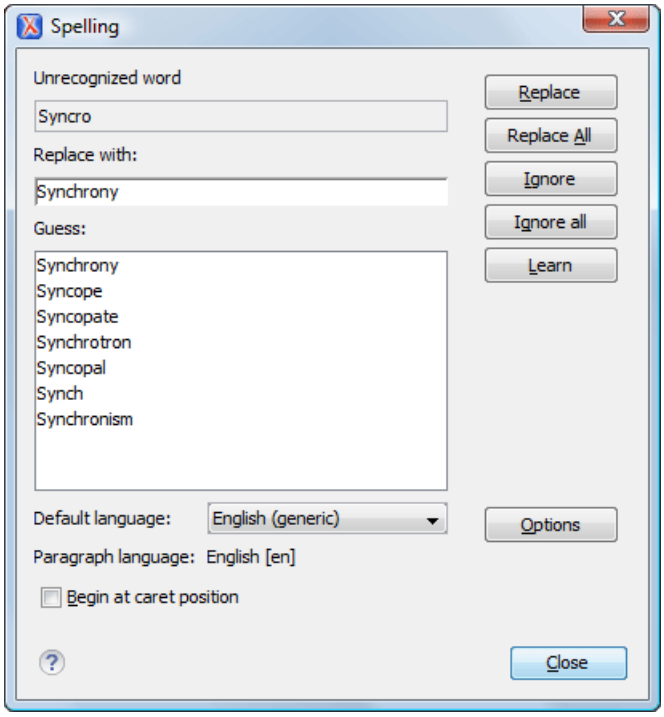


Figure 352: The Check Spelling Dialog

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 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 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: **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.

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 comes with the following built-in dictionaries for the Hunspell checker:

- English (US)
- English (UK)
- French
- German (old orthography)
- German (new orthography)
- Spanish

There is one dictionary for each language-country variant combination. If you cannot find a Hunspell dictionary that is already built for your language you can build such a dictionary. First you need a list of correct words which you want to check in your documents. You build a dictionary from this list following *these instructions*.

Adding a Dictionary for the Hunspell Checker

To add a new spelling dictionary to Oxygen or to replace an existing one you should follow these steps:

1. *Download the archive* with the files of your language dictionary.

   A dictionary has two files with the same name and different extensions: a file with `.dic` extension and a file with `.aff` extension.
2. Copy the `.aff` and `.dic` files to the `spell` subfolder (`.spell` subfolder on Linux and Mac OS X) of the Oxygen preferences folder only if it is a new dictionary (it is not available as built-in dictionary in Oxygen).

   The Oxygen preferences folder is `[APPLICATION-DATA-FOLDER]/com.oxygenxml`, where `[APPLICATION-DATA-FOLDER]` is:

   - `C:\Documents and Settings\[LOGIN-USER-NAME]\Application Data` on Windows XP
   - C:\Users\[LOGIN-USER-NAME]\AppData\Roaming on Windows Vista
   - `[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` only if it is an existing dictionary.
4. Restart the application after copying the dictionary files.

Dictionaries for the Java Checker

A Java checker dictionary has the form of a `.dar` file located in the directory `[Oxygen-install-folder]/dicts`. Oxygen 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. There is one dictionary for each language-country variant combination. If you cannot find a dictionary that is already built for your language you can build such a dictionary with the tool available at *http://www.xmlmind.com/spellchecker/dictbuilder.shtml*.

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 folder:

- `[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

To delete items from the list of learned words, press **Delete learned words** from *Spell Check* 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 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:

- **Delete Repeated Word** action - allows you to delete repeated words;

- a list of words suggested by the spell checking engine as possible replacements of the unknown word;
- **Learn Word** action - allows you to add the current unknown word to the persistent dictionary.

Spell Checking in Multiple Files

The **Check Spelling in Files** action available from the **Edit** menu or from the **Project** contextual menu enables you to check spelling on multiple documents.

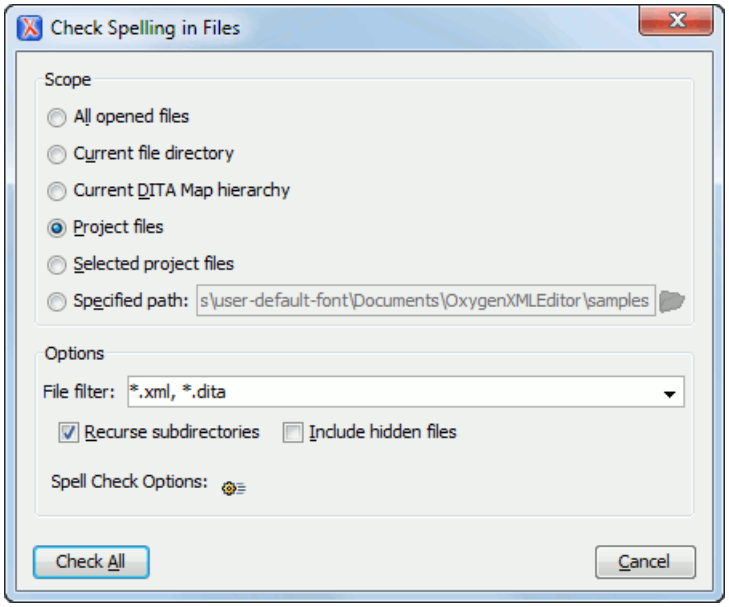


Figure 353: Check Spelling in Files Dialog

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.
- **Scope of the current DITA Map** - All the files referred in the current DITA map opened in the **DITA Maps Manager** view.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - A custom path.

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* .

Changing the Font Size

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

- **Document** > **Font size** > **Increase editor font (Ctrl + NumPad + +)** - Increases the font size with one point for each execution of the action.
- **Document** > **Font size** > **Decrease editor font (Ctrl + NumPad + -)** - Decreases the font size with one point for each execution of the action.
- **Document** > **Font size** > **Normal editor font (Ctrl + 0)** - Resets the font size to *the value of the editor font set in Preferences.*

Word/Line Editor Actions

The Text editor implements the following actions:

- **(Ctrl+Delete (Meta+Delete on Mac))** - Deletes the next word.
- **(Ctrl+Backspace (Meta+Backspace on Mac))** - Deletes the previous word.
- **(Ctrl+W (Meta+W on Mac))** - Cuts the previous word.
- **(Ctrl+K (Meta+K on Mac))** - Cuts to end of line.

Dragging and Dropping the Selected Text

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

Inserting a File at Caret Position

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

Opening the File at Caret in System Application

The action from menu **Document** > **File** > **Open File at Caret in System Application** opens the filename under the current position of the caret with the default system application associated with the file.

Opening the File at Caret Position

The action from menu **Document** > **File** > **Open File at Caret** opens in a new panel the file with the file path at the current position of the caret. If the file does not exist at the specified location the error dialog that is displayed contains a **Create new file** button which starts the **New document** wizard. This allows you to choose the type or the template for the file. If the action succeeds, the file is created with the 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 caret position.

Switching Between Opened Tabs

There are two actions for cycling through the opened file tabs:

- **(Ctrl + Tab)** - Switches between the tabs with opened files in the order most recent ones first.
- **(Ctrl + Shift + Tab)** - Switches between the tabs with opened files in the reverse order.

Printing a File

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

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

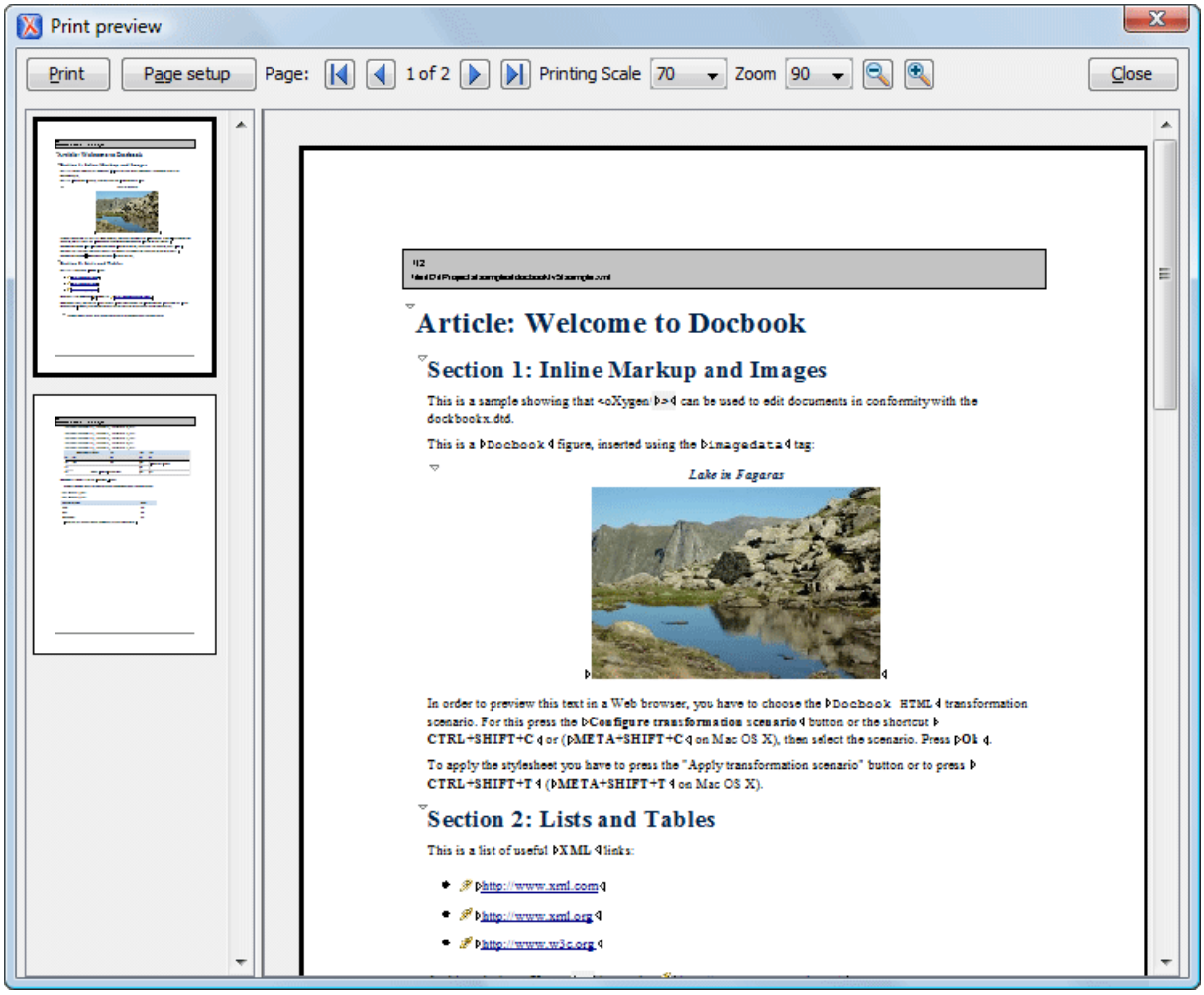


Figure 354: Print Preview Dialog

The main window is split in three sections:

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

Exiting the Application

The action from menu **File** > **Exit (Ctrl+Q)** terminates Oxygen . Session information such as the current project, open documents and user preferences is made persistent. When Oxygen is re-opened, this persistent information returns to the last saved state.

Chapter 24

Configuring the Application

Topics:

- *Importing / Exporting Global Options*
- *Preferences*
- *Sharing Preferences*
- *Automatically Importing the Preferences from the Other Distribution*
- *Reset Global Options*
- *Scenarios Management*
- *Editor Variables*
- *Configure Toolbars*

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.

Importing / Exporting Global Options

In the **Options** menu you can find the import / export preferences operations which allow you to move your global preferences in XML format from one computer to another.

Preferences

Once the application is installed you can use the **Preferences** dialog accessed from menu **Options** > **Preferences** to customize the application settings for your requirements and network environment.

There is a search field available in the dialog for selecting only the preferences panels containing required words in the panel title or in the text of labels, buttons, tables, etc contained in the panel. If you want to go to first match press **(Enter)**, **(Up Arrow)** or **(Down Arrow)**.

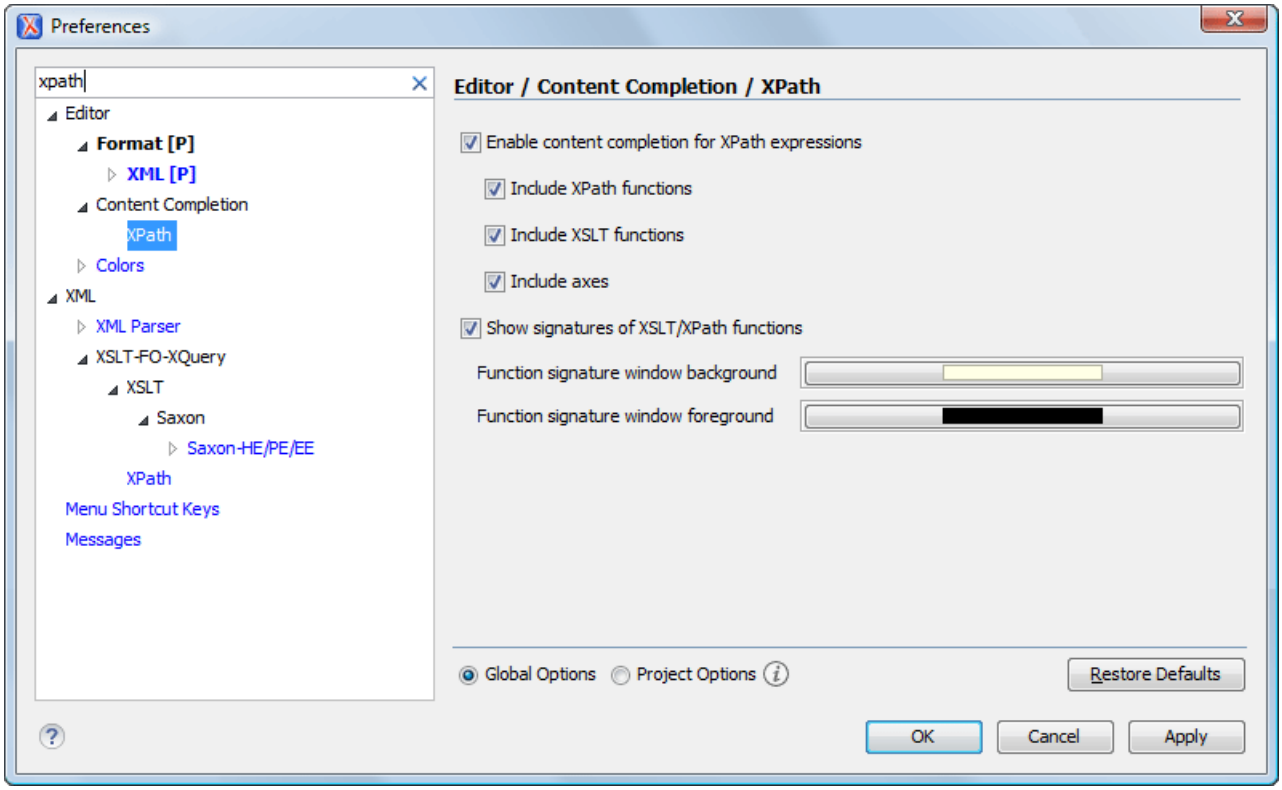


Figure 355: The Search field from the Preferences dialog

You can always revert modifications to their default values by pressing the **Restore Defaults** button, available in each preference 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.

Global

The **Global** preferences panel is opened from menu **Options** > **Preferences** > **Global** .

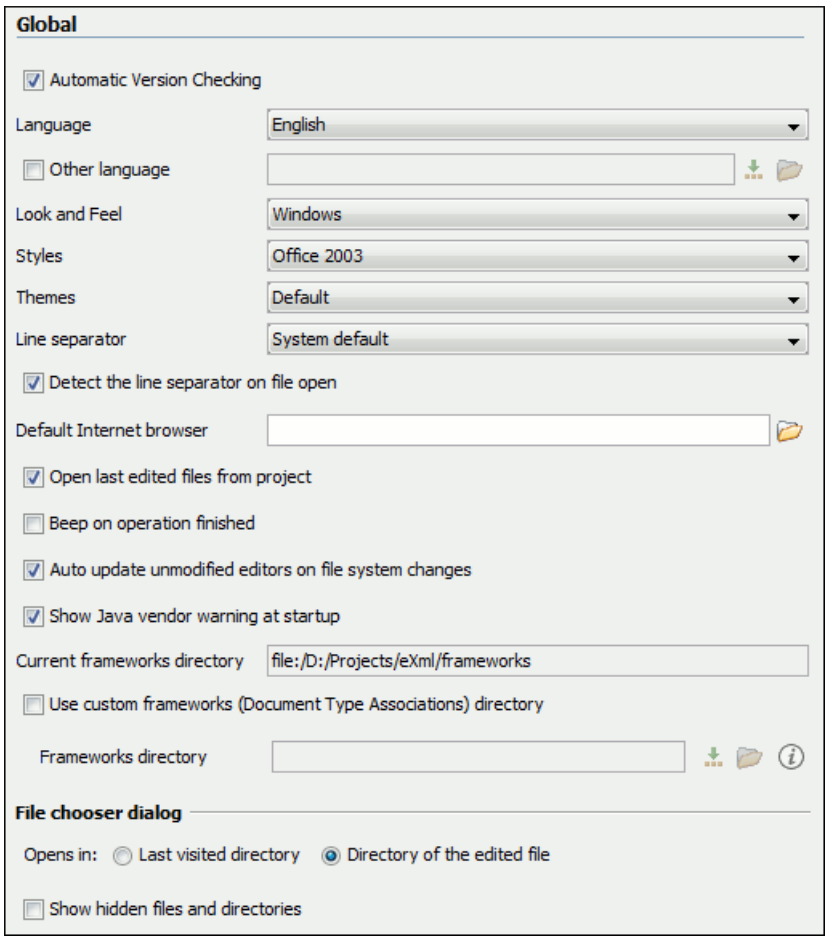


Figure 356: The Global preferences panel

The following user preferences are av available in this panel:

- **Automatic Version Checking** - When enabled, checks the availability of new Oxygen versions at *http://www.oxygenxml.com* .
- **Language** - The application supports a number of languages for localization of the GUI. Go to menu **Options** > **Preferences** > **Global** and select the **Language** drop-down list to display the language choices.

  Note: After restarting the application, if some GUI labels are not rendered correctly (for example Chinese or Korean characters) you will need to install the corresponding language pack from your OS installation kit (for example the East-Asian language pack).

- **Other language** - To change the user interface language of Oxygen you must set here the properties file with all the user interface messages and labels translated to your preferred language. For details about creating this file see *the section describing the creation process*. After setting the file you have to restart Oxygen in order to change the user interface language to your preferred language.
- **Look and Feel** - Use this option to change graphic style (look and feel) of the GUI.
- **Styles** - On Windows there are available the following styles:
  - Office 2003
  - Vsnet
  - Eclipse
  - Xerto
  - Default


Note: After changing the style you have to restart the application in order for the modification to take effect.

On Linux there are available the following styles:

- Eclipse
- Default


Note: After changing the style you have to restart the application in order for the modification to take effect.

On Mac OS X this option is not available.

- **Themes** - On Windows this option is enabled only for the **Office 2003** and **Default** styles. In these cases, the following themes are available:
  - Normal Color
  - Home Stead
  - Metallic
  - Default
  - Gray

  On Linux and Mac OS X this option is not available.
- **Line separator** - This option defines the line separator. The **System Default** choice sets the line separator of the platform.
- **Detect the line separator on file open** - When this option is checked the editor will detect the line separator when the edited file is loaded and it will use it when the file is saved. The new files are saved using the line separator defined by the **Line separator** option.
- **Default Internet browser** - The path to a web browser executable. The browser is used for:
  - opening (X)HTML or PDF transformation results
  - opening the Oxygen home page
  - pointing to specific paragraphs in the W3C recommendation of XML Schema on the W3C website in case of XML Schema validation errors
- **Open last edited files from project** - When enabled, Oxygen will open the last edited files from project at start-up.
- **Beep on operation finished** - If checked, it notifies the user through a beep that an action has ended. It will notify the user only at the end of validate, check well-formedness and transform actions.
- **Show Java vendor warning at startup** - Sun Microsystems/Oracle Java VM (on Windows and Linux) or Apple Computer Java VM (on Mac OS X) is recommended for running Oxygen . If a different VM is used, then a warning is displayed. This option allows the user to choose whether the warning dialog is shown or not.
- **Use custom frameworks directory** - For editing different types of XML documents (for content completion, validation, authoring) Oxygen can use information from the document types which are stored in the `frameworks` subfolder of the Oxygen install folder. If a custom `frameworks` folder is specified then Oxygen will load the document types from this location.
- **Auto update unmodified editors on file system changes** - Checked by default. If checked, the synchronization of the unmodified (not dirty) editors with the system changes is done automatically, without the user's interaction.
- **Last visited directory** - The *dialog for opening files* remembers the last visited folder and the next time it starts directly in this folder.
- **Directory of the edited file** - The *dialog for opening files* starts in the folder where the currently edited file is stored.
- **Show hidden files and directories** - Shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on Mac OS X.

Fonts

The **Fonts** preferences panel is opened from menu **Options** > **Preferences** > **Fonts** .

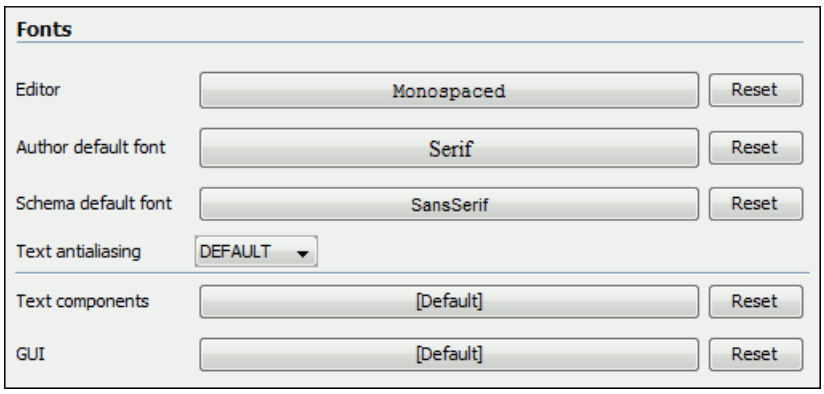


Figure 357: The Fonts preferences panel

The fonts that can be configured in Oxygen are the following:

- **Editor** - The font family and font size used to display text in the editor. This option affects both the text and grid page of the editor.
- **Author default font** - The font family and font size used to display text in Author mode of the XML editor. This value is used as default in case another one is not specified in the CSS associated with the opened document.
- **Schema default font** - The font family and font size used for displaying text in:
  - the **Design** mode of the W3C XML Schema editor (the schema diagram);
  - images with schema diagram fragments that are included in the HTML documentation generated from an XML Schema.
- **Text antialiasing** - Enables text anti-aliasing at the specified level. On JVM versions before 1.6 this combo box contains only the values **Default**, **On** and **Off**. Default means that Oxygen does not set anything special for text anti-aliasing but the JVM uses the setting of the operating system, if available. The **On** option sets the text anti-aliasing to pixel level and the **Off** option disables it. Starting with version 1.6 the combo contains also values specific for sub-pixel anti-aliasing, like GASP, LCD_HRGB, LCD_VRGB which sets the respective anti-aliasing mode for the text displayed in the Oxygen editors and views.
- **Text components** - The font family and font size used to display text in text components. After changing the font, restart the application to see the effect.
- **GUI** - The font family and font size used to display GUI labels. After changing the font, restart the application to see the effect.

Document Type Association

The **Document Type Association** preferences panel is opened from menu **Options** > **Preferences** > **Document Type Association** .

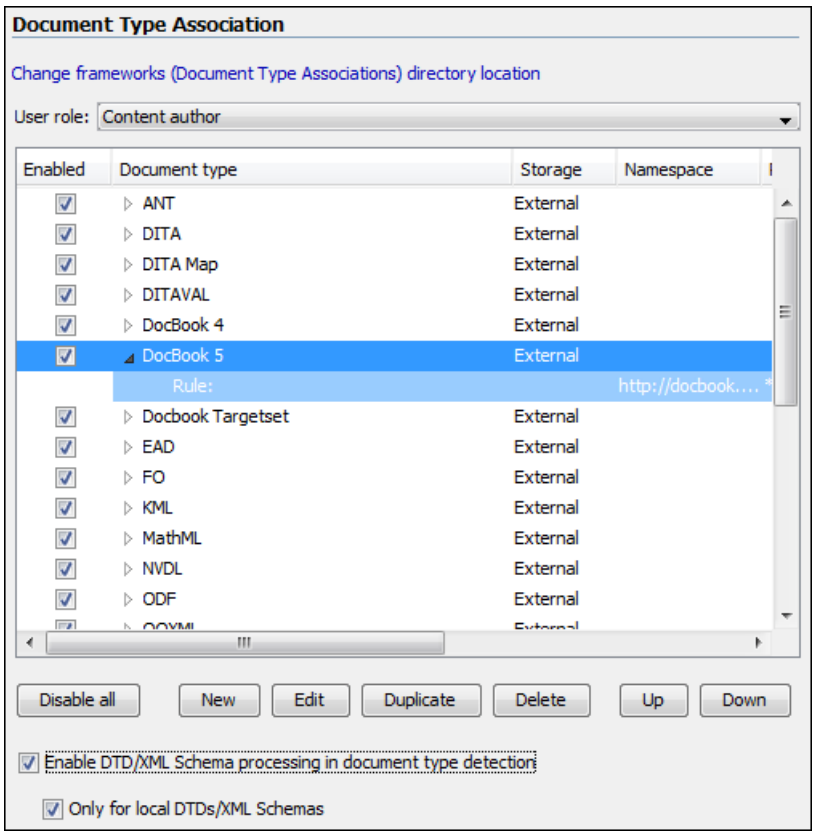


Figure 358: Document Type Association preferences panel

The following actions are available in this preferences panel:

- **Change framework directory location** - Specifies a *custom frameworks folder* from where Oxygen loads the document types.
- **User roles** - Selects between two user roles: **Content author** and **Developer**. When the selected role is **Content author**, you can modify only the properties of the **Document Type Associations** stored in the user preferences. The externally stored associations (that means all the built-in document types) cannot be modified and you must duplicate them to further customize these associations. The **Developer** user can change any document type association.
- **Document types table** - Presents the currently defined document type associations. The columns are:
  - **Document type** - Contains the name of the document type.
  - **Enabled** - When checked, the corresponding document type association is enabled. The association is analyzed when trying to determine the type of a document opened in Oxygen.
  - **Storage** - Displays the type of location where the framework configuration file is stored.

    Note: Please note that if the document type association settings are already stored in a framework file, the file will be removed and its content will be saved in Oxygen internal options.

  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 class which is used to determine if a document matches the rule.

- **New** - Opens a dialog that allows you to add a new association. New association is added to top of document type list.
- **Edit** - Opens a new dialog allowing you to edit an existing association.
- **Delete** - Deletes one of the existing association.
- **Up** - Moves the selected association one level up (the order is important because the first document type association in the list that matches the document will be used).
- **Down** - Moves the selected association one level down.
- **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 analyzed also, if this is specified in the association rules.

  If you are writing DITA customizations, it is recommended to keep this checkbox enabled. DITA topics and maps are also matched by looking for the `DITAArchVersion` attribute in the root element. If the DTD is not processed on detection, then this attribute specified as `default` in the DTD will not be detected on the root element and the DITA customization will not be correctly matched.

  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.

Perspectives Layout

The **Perspectives Layout** preferences panel is opened from menu **Options** > **Preferences** > **Perspectives Layout** .

Oxygen has a large number of helper views that can be arranged in different layouts. Use this preferences panel to configure the layout.

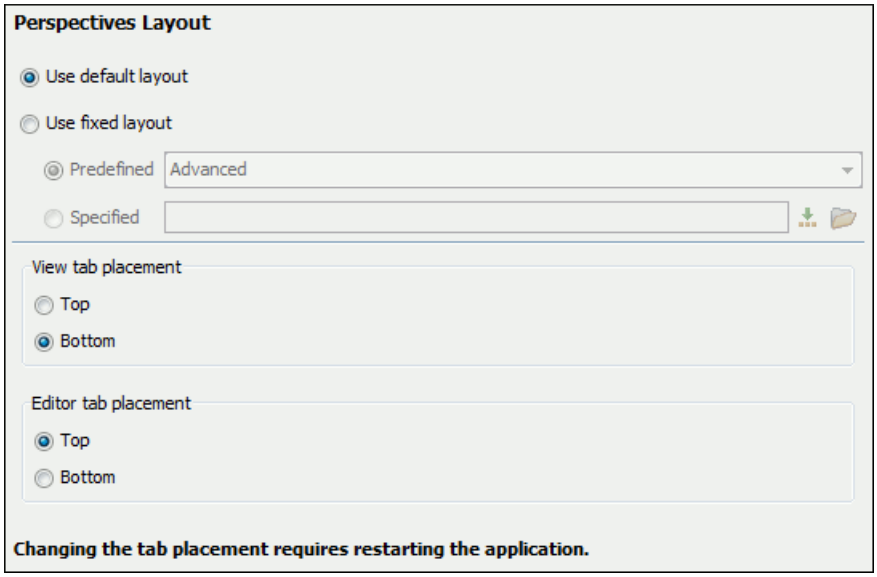


Figure 359: The Perspectives Layout Preferences Panel

The following preferences are available:

- **Use default layout** - Selected by default. It indicates that the editor must use the default layout for all the perspectives. Any modification of this layout (for instance closing / showing views or a new view arrangement) is saved when the program exits and is reloaded at the next start up.
- **Use fixed layout** - Check this option when you want the editor to always start with a certain layout. Modifications of the selected layout are lost when the program exits. There are two kinds of fixed layout:

- **Predefined** - Oxygen has several predefined layouts to choose from, depending on the type of work you intend to do:
  - **Advanced** - All the views are visible.
  - **Basic** - Just the **Project** view and the **Outline** view are visible. This is recommended when you edit XML content and you need screen space.
  - **Intermediate** - The **Project**, **Outline**, **Attributes** and **Model** views are visible.
  - **Schema development** - The **Project**, **Outline**, **Attributes**, **Model** views and **Schema Components** are visible.
  - **XQuery development** - Only the **Project** view and the editing area are visible.
  - **XSLT development** - The **Project**, **Outline**, **Attributes**, **Model** and **XSLT Input** views are visible.
- **Specified** - You can choose an existing layout file from disk. In order to create such a file, you can arrange the views in the desired order and then save it with the action from menu **Window** > **Save layout ...**

Encoding

The **Encoding** preferences panel is opened from menu **Options** > **Preferences** > **Encoding**

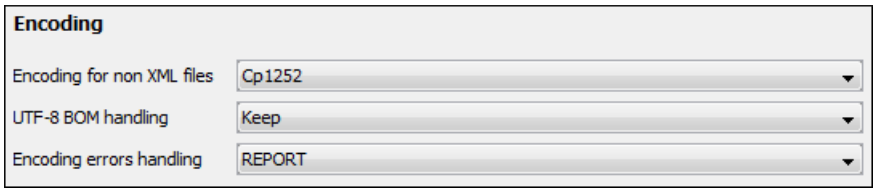


Figure 360: The Encoding preferences panel

The encoding preferences are the following:

- **Encoding for non XML files** - The default encoding to be used when opening non XML documents. This is necessary because non XML files have a large variety of formats and there is no standard mechanism for declaring the encoding that should be used for opening and saving the file. In case of XML files the encoding is usually declared at the beginning of the file in a special declaration or it assumes the default value UTF-8.
- **UTF-8 BOM handling** - Specifies how to handle the BOM (Byte Order Mark) for UTF-8 XML documents on the document save action. The UTF-16 BOM is always preserved. In case of UTF-32 documents the BOM is for big endian.

  The available BOM handling policies are:
  - **Don't Write** - Don't write the BOM bytes, the loaded BOM bytes are ignored;
  - **Write** - Write the BOM bytes accordingly with chosen encoding;
  - **Keep** - If the loaded document has BOM then write them accordingly with chosen encoding. This is the default option.
- **Encoding errors handling** - This option defines how to handle characters that cannot be represented in the specified encoding of the document when the document is opened. The available options are:
  - **REPORT** - Shows an error dialog with the character that cannot be represented in the specified encoding and allows the user to decide how to continue (ignore that character, replace it with a standard replacement character). This is the default option.
  - **IGNORE** - The character is ignored and it will not be included in the document displayed in the editor panel.
  - **REPLACE** - Replace the character with a standard replacement character. For example if the encoding is UTF-8 the replacement character has the Unicode code `FFFD`, and if the encoding is ASCII the character code is 63.

Editor

The **Editor** preferences panel is opened from menu **Options** > **Preferences** > **Editor** .

Use these options to configure the visual aspect of the text editor.

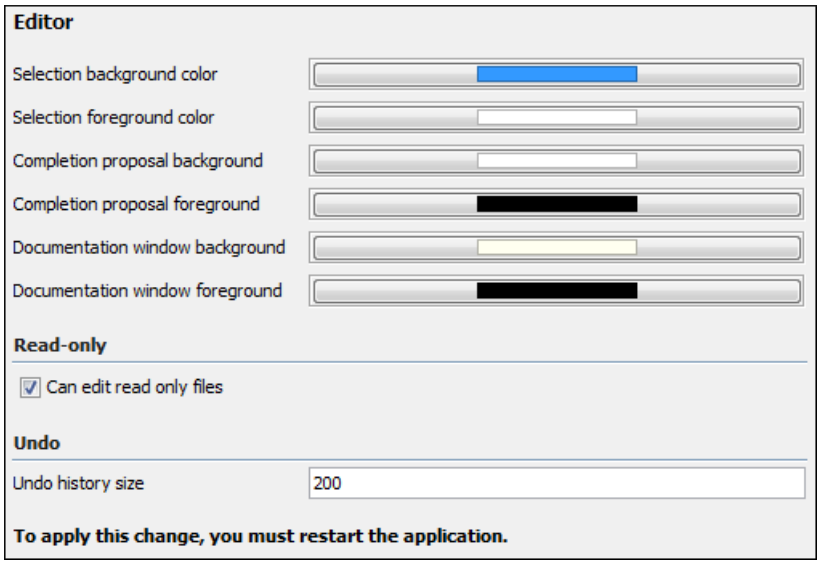


Figure 361: The Editor Preferences Panel

The following options are available in this panel:

- **Selection background color** - Background color of selected text.
- **Selection foreground color** - Text color of selected text.
- **Completion proposal background** - Background color for the content completion window.
- **Completion proposal foreground** - Foreground color for the content completion window.
- **Documentation window background** - Background color for the window containing documentation for the content completion elements.
- **Documentation window foreground** - Foreground color for the window containing documentation for the content completion elements.
- **Can edit read only files** - Read-only files are marked in Oxygen using a lock icon on the file tab. If this option is checked, you are able to modify a read-only file, but you cannot overwrite it when trying to save the file. If unchecked, any modification to the content is prohibited.
- **Undo history size** - Sets the maximum amount of undo edits which is remembered by the editor in each of the pages (Text, Author, Design, Grid).

Pages

The **Pages** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** and allows you to select the initial page for an editor. The mode in which a file was edited in the previous session is saved and will be used when the application is restarted and the file reopened.

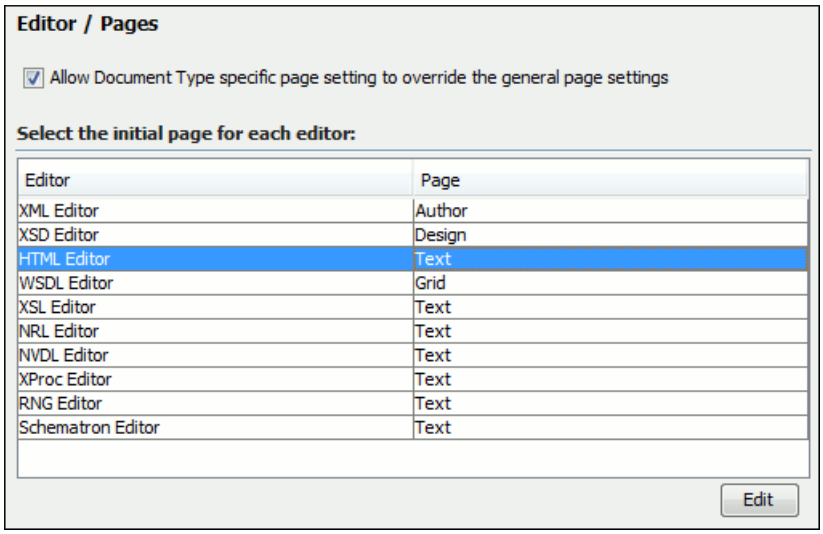
If the checkbox **Allow Document Type specific page setting to override the general page setting** is checked the initial page setting from *the Document Type dialog* will override the initial page setting from the table that is explained below.

The initial page of each editor type has one of the following values:

- Text
- Author
- Grid

- Design (available only for the W3C XML Schema editor)

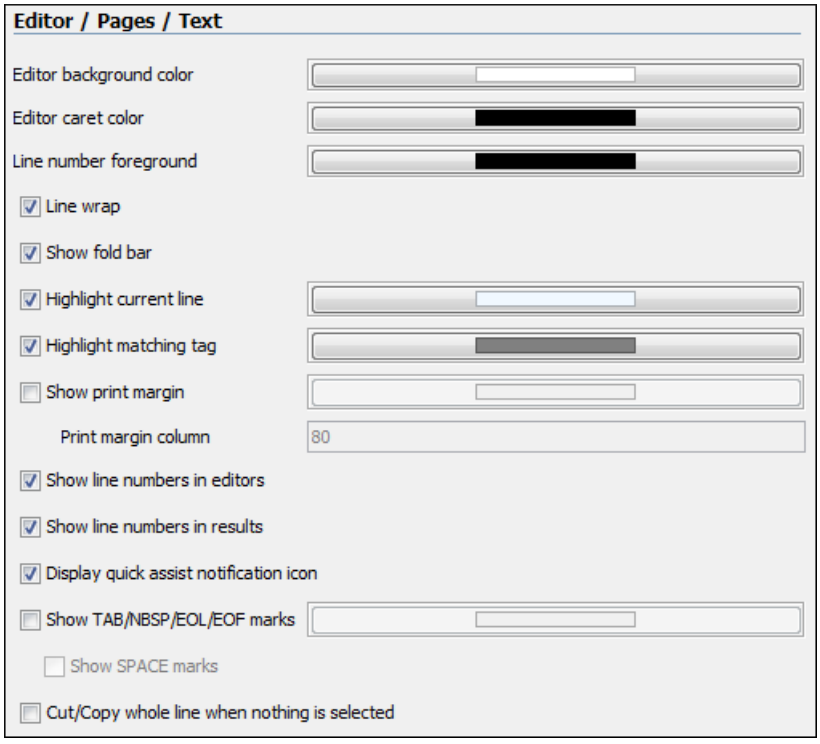
The Oxygen Pages Preferences Panel



Text

The **Author** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Text** .

The Oxygen Text Preferences Panel



The following preferences are available:

- **Editor background color** - Background color of the editor and also of the Diff Files' editors.
- **Editor caret color** - Customize the caret color.
- **Line number foreground** - Foreground color for the line numbers displayed at the left of editor panel.
- **Line wrap** - Enable soft wrap of long lines, that is automatically wrap lines in edited documents without inserting newline characters in the file.
- **Show fold bar** - Enables the display of the document folding bar.

- **Highlight current line** - Enables highlight for the current line. Use the button to set the highlight color.
- **Highlight matching tag** - Enables highlight for the tag matching the one on which the caret is situated. Use the button to set the color of the highlight.
- **Show print margin** - If the current content is printed with the **File** > **Print** action, this action enables displaying a vertical line in the editor panel representing the paper margin. Use the button to set the color of the print margin line.
- **Print margin column** - The number of characters included on a line which the print format allows.
- **Show line numbers in editor** - Enables the line numbers column located in the left part of the editing space. When unchecked, line numbers option is disabled.
- **Show line numbers in results** - Enables the line numbers column located in the left part of the results panel in the Debugger perspective.
- **Show TAB/NBSP/EOL/EOF marks** - Marks the TAB/NBSP/EOL/EOF using small icons, for a better visualization of the document. Also set the marks color.
- **Display quick assist notification icon** - Displays the *Quick Assist* yellow bulb icon in the editor line number stripe.
- **Cut / Copy whole line when nothing is selected** - Enables the Cut / Copy shortcut keys when nothing is selected in the editor. The Cut / Copy actions operate on the entire current line.

Text / Diagram

If operation is slow for very large schemas disabling the schema diagram view will improve the speed of navigation through the edited schema.

The **Diagram** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Text / Diagram** .

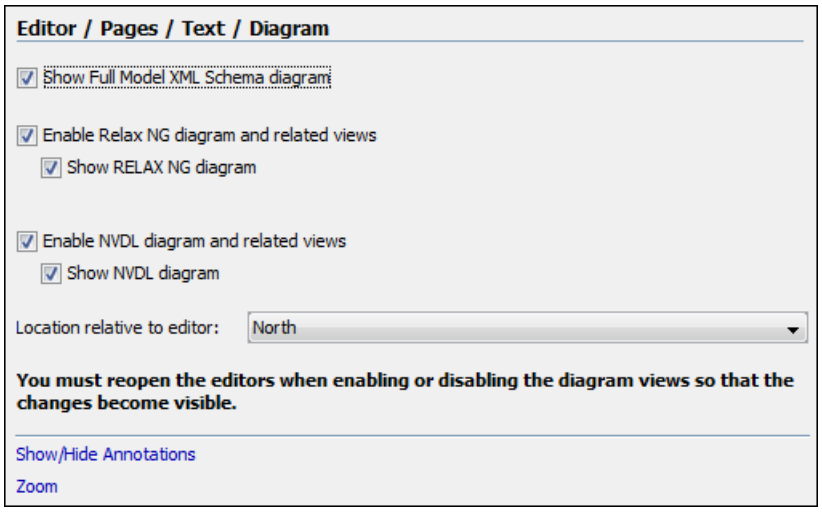


Figure 362: Schema Diagram Preferences Panel

- **Show Full Model XML Schema diagram** - If this option is enabled the *old synchronized version* of the schema diagram will be available in the Text page for XML Schemas. For editing in the schema diagram, you can use the *new schema diagram* editor page.
- **Enable Relax NG diagram and related views** - If this option is disabled the schema diagram for Relax NG will not be generated and displayed, also the related views that present the schema components are not populated with data. In case you need the related views to be active, you can let this option checked and uncheck the following one.
- **Show Relax NG diagram** - If this option is disabled the schema diagram for Relax NG schemas will not be generated and displayed.
- **Enable NVDL diagram and related views** - If this option is disabled the schema diagram for NVDL will not be generated and displayed, also the related views that present the schema components are not populated with data. In case you need the related views to be active, you can let this option checked and uncheck the following one.
- **Show NVDL diagram** - If this option is disabled the schema diagram for NVDL schemas will not be generated and displayed.

- **Location relative to editor** - The location of the diagram panel in the editing area can be: North, East, South, West. For example North means that the diagram panel takes the North part of the editing area and the text editor panel takes the rest of the editing area.

Grid

The **Grid** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Grid** .

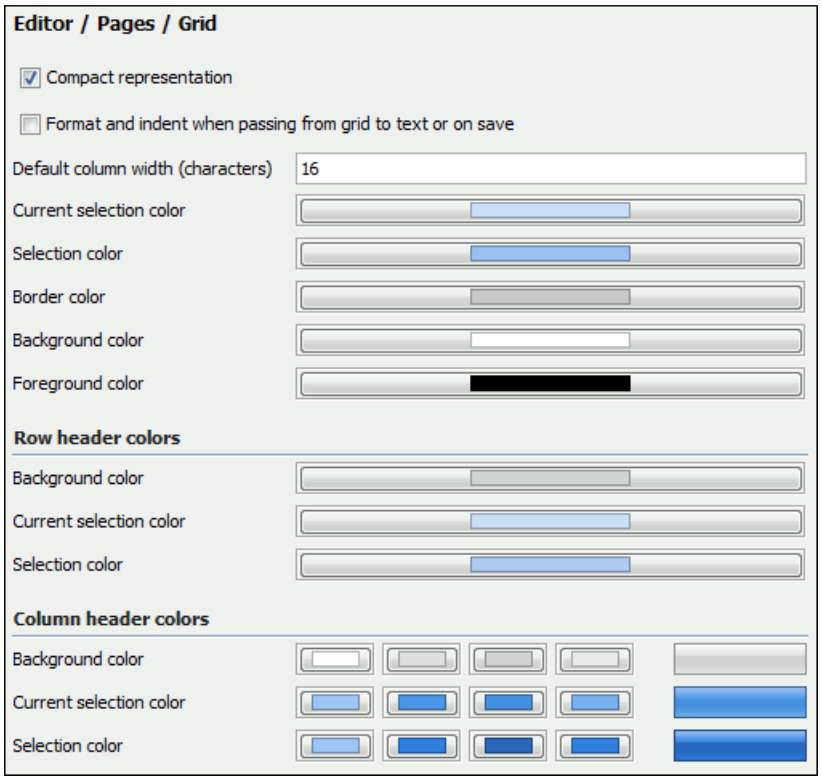


Figure 363: The Grid Editor Preferences Panel

The following preferences are available for Grid mode of the XML editor:

- **Compact representation** - If checked a child element is displayed at the same height level with the parent element. If unchecked a child elements is presented nested with one level in the parent container, that is lower than the parent with one row.
- **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 an element name and its text content, an attribute name and its value. If the total width of the grid structure is too large you can resize any column with the mouse but the change is not persistent. To make it persistent set the new column width in this user option.
- **Current selection color** - Background color used in the focused selected cell of the grid to make it different in the set of selected cells. For example when an entire row is selected only one cell of the row is the focused selected one.
- **Selection color** - Background color used in the selected cells of the grid except the focused selected cell which uses a different background color.
- **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 color of the text used for the element names, text content of elements, attribute names and attribute values.
- **Row header colors - Background color** - The background color of row headers that are not selected.
- **Row header colors - Current selection color** - The background color of the row header that is currently selected and has the focus.

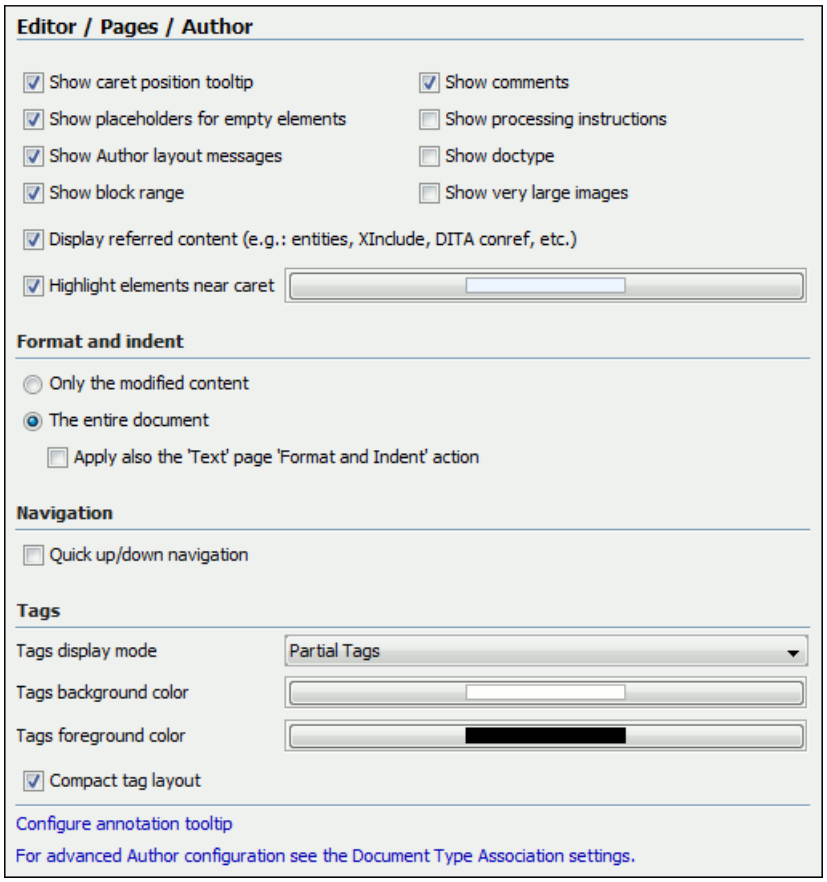
- **Row header colors - Selection color** - The background color of the row header that is currently selected and does not have the focus.
- **Column header colors - Background color** - The background color of column headers that are not selected.
- **Column header colors - Current selection color** - The background color of the column header that is currently selected and has the focus.
- **Column header colors - Selection color** - The background color of the column header that is currently selected and does not have the focus.

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

The **Author** preferences panel is opened from menu **Options** > **Preferences** > **Pages** > **Editor** > **Author** .

The Oxygen Author Preferences Panel



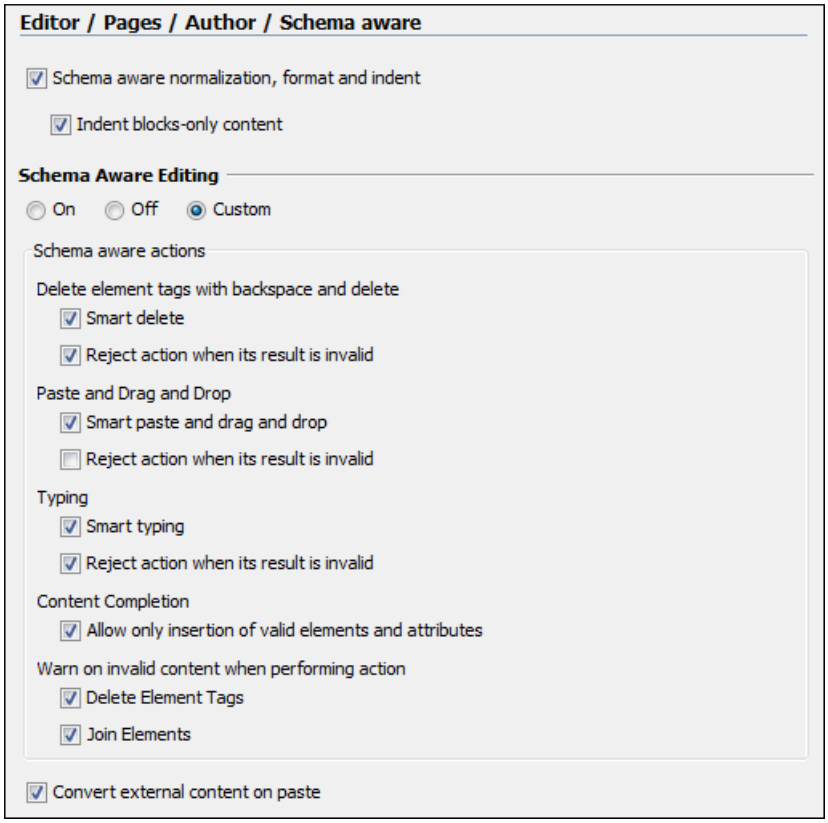
- **Show caret position tooltip** - If checked, the position information tooltip is displayed. More information about the position information tooltip can be found in the section *Position information tooltip*. The documentation tooltip can be disabled from the *Content Completion Annotations preferences panel*.
- **Show placeholders for empty elements** - When checked, placeholders are displayed for empty elements to make them clearly visible.
- **Show Author layout messages** - If checked, all errors reported during layout creation is presented in the **Errors** view.
- **Show block range** - If checked, a block range indicator is shown in a stripe located in the left side of the editor.
- **Hide comments** - When checked, comments from the documents edited in Author mode are hidden.
- **Hide processing instructions** - When checked, processing instructions from the documents edited in Author mode are hidden.
- **Hide doctype** - When checked, doctype sections from the documents edited in Author mode are hidden.

- **Show very large images** - If unchecked, images larger than 6 megapixels (24MB uncompressed) are not loaded and displayed in Author mode. Please be aware that this option is unchecked by default because of the large amounts of application memory that images of high resolution can occupy. As a result, an OutOfMemory error could occur which would practically make Oxygen unusable without a restart of the entire application.
- **Display referred content (for example entities, XInclude, DITA conref, etc.)** - When checked, the references (entities, XInclude, DITA conref, etc) also display the content of the resources they refer.
- **Highlight elements near caret** - Background color of the element or elements at cursor position.
- **Format and indent** - Here you can set the method of format and indent that is applied when a document is saved in Author mode:
  - **Only the modified content** - The save operation formats only the nodes that were modified in Author mode.
  - **The entire document** - The save operation applies formatting to the entire document regardless of the nodes that were modified in Author mode. If the checkbox **Apply also the 'Text' page 'Format and Indent' action** is selected, the content of the document is formatted by applying the **Format and Indent** action on every switch from the Author editor to the Text editor of the same document.
- **Quick up / down navigation** - Speeds up navigation between blocks when using up / down keys. The cursor stops on the next / previous line.
- **Tags display mode** - Default display mode for element tags presented in **Author** mode. You can choose between:
  - **Full Tags with Attributes**
  - **Full Tags**
  - **Block Tags**
  - **Inline Tags**
  - **Partial Tags**
  - **No Tags**
- **Tags background color** - Allows you to configure the Author tags background color.
- **Tags foreground color** - Allows you to configure the Author tags foreground color.

Schema Aware

The **Schema Aware** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Author** > **Schema aware** .

The Oxygen Schema Aware Preferences Panel



- **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 ***Options > Preferences > 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 will take into account the schema.
  - **On** - Enables all schema aware editing options.
  - **Off** - Disables all schema aware editing options.
  - **Custom** -
    - **Delete element tags with backspace and delete** - Controls the behaviour 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:
    - 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.
    - You will iterate 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).
  - **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 behaviour 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 behaviour 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 form the content completion proposals list can be inserted in the document through content completion.
- **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** - when checked, Oxygen 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).

If the Schema Aware Editing is **On** or **Custom** all actions that can generate invalid content will be forwarded first toward *AuthorSchemaAwareEditingHandler*.

Review

The Author **Review** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Author** > **Review** .

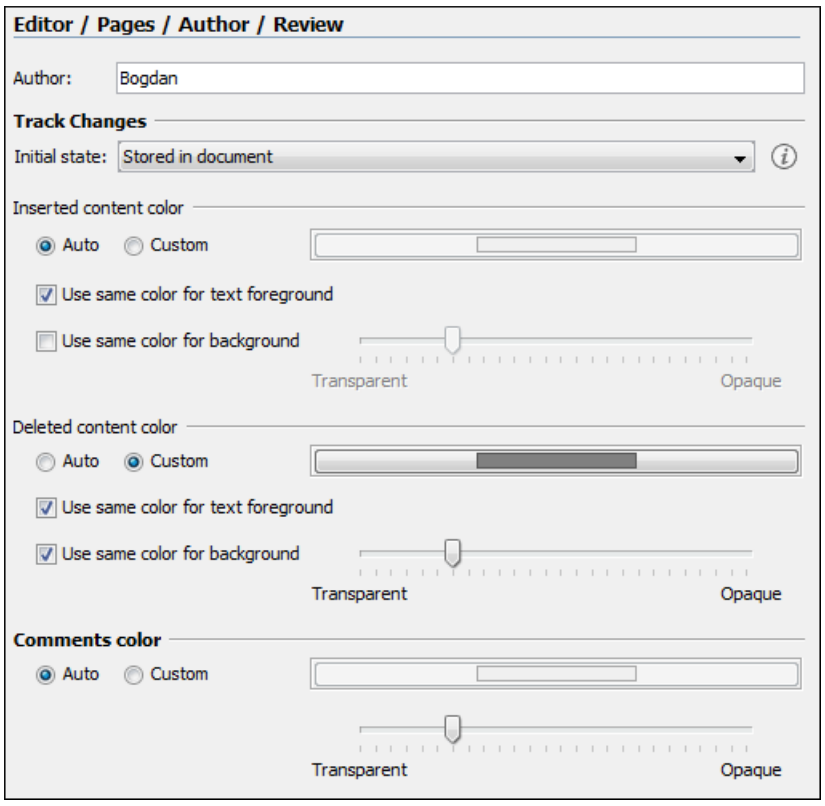


Figure 364: The Oxygen Review Preferences Panel

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 will be associated with each performed change.
- **Initial Track Changes 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, a processing instruction saved in the document decides to enable or disable the **Track Changes**. When this option is enabled and you open a document in Oxygen Author, 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 Oxygen Author.
  - **Always On** - The **Track Changes** is active every time you open a document.
  - **Always Off** - The **Track Changes** is inactive every time you open a document.

  👉 **Note: Initial Track Changes State** options do not affect documents already open in Oxygen Author.

- **Inserted content color** -
  - **Auto** - Automatically assign colors for the insert changes based on the author name.
  - **Custom** - Use a custom color for all insert changes, regardless of the author name.
  - **Use same color for text foreground** - Use the same color to paint the inserted text foreground.
  - **Use same color for background** - Use the same color for the insert text background with a certain transparency.
- **Deleted content color** -
  - **Auto** - Automatically assign colors for the delete changes based on the author name.
  - **Custom** - Use a custom color for all delete changes, regardless of the author name.
  - **Use same color for text foreground** - Use the same color to paint the deleted text foreground.

- **Use same color for background** - Use the same color for the delete text background with a certain transparency.

- **Commented content color** -
  - **Auto** - Automatically assign colors for the commented content based on the author name.
  - **Custom** - Use a custom color for all commented content, regardless of the author name.

Profiling / Conditional Text

The Author **Review** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Author** > **Profiling/Conditional Text** .

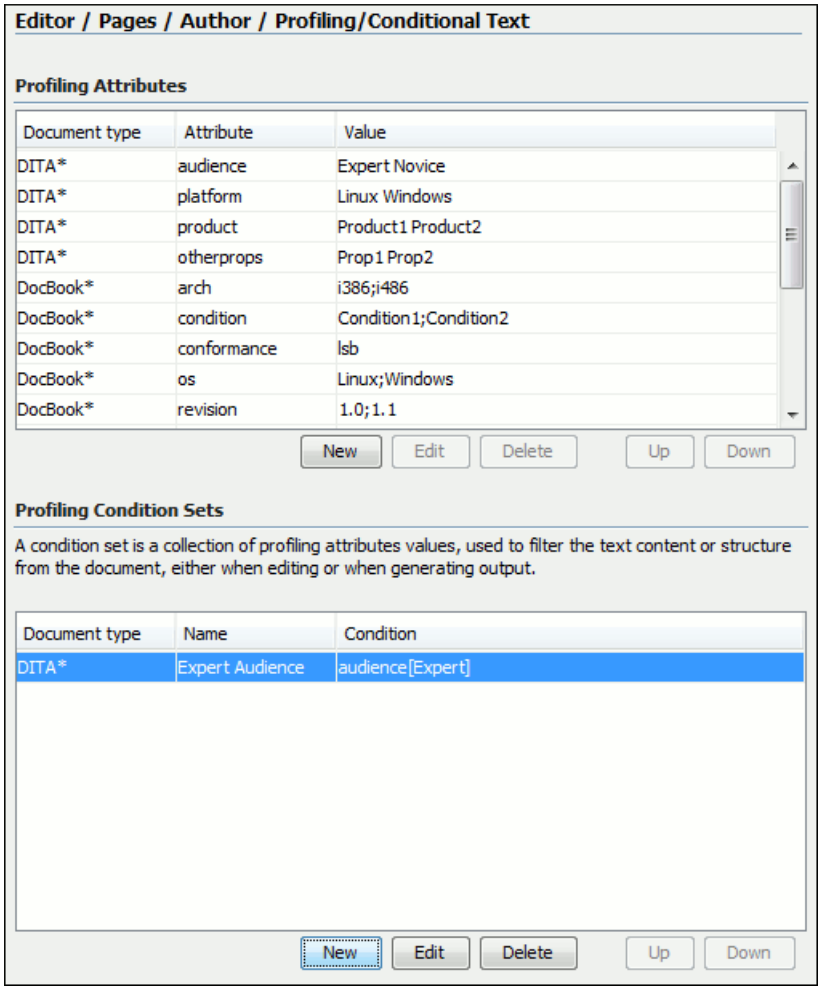


Figure 365: Profiling / Conditional Text Preferences Panel

Here you can define, edit, delete new profiling attributes and profiling condition sets.

MathML

The MathML editor of Author mode has the following configurable parameters:

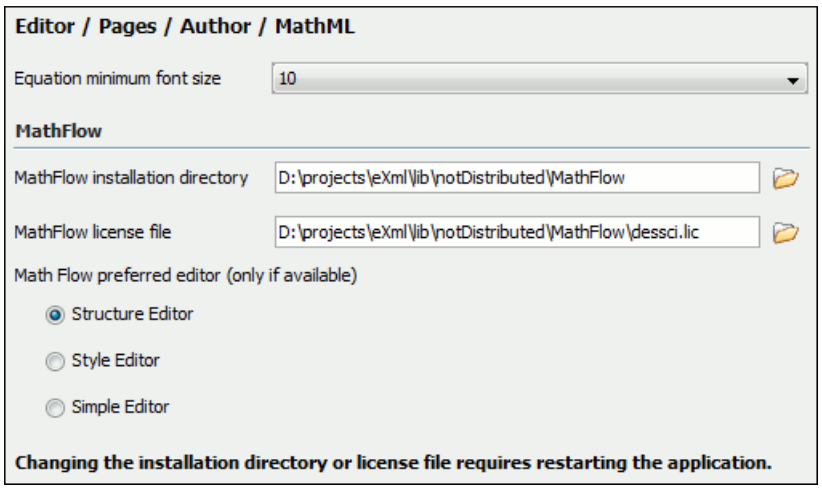


Figure 366: Author MathML editor preferences panel

- **Equation minimum font size** - The minimum size of the font used for rendering mathematical symbols.
- **MathFlow installation directory** - The installation folder where Oxygen will find and load 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 MathFlow Components (MathFlow SDK): structure editor, style editor and simple editor. More documentation is available on the website of *MathFlow SDK*.

Messages

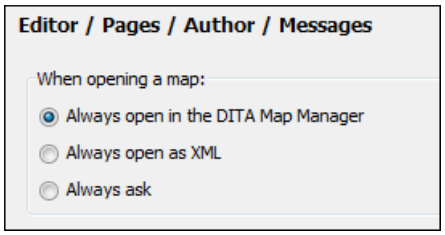


Figure 367: The Author's Messages preferences panel

The following preferences are applied to the warning or notification messages that Author mode displays for the content author:

- **When opening a map** - Specifies the default behavior when trying to open a map. You can choose between:
  - **Always open in the DITA Map Manager** - A DITA map file will be opened always in the DITA Maps Manager view without asking a confirmation from the user.
  - **Always open as XML** - A DITA map file will be opened always in the XML editor panel without asking a confirmation from the user.
  - **Always ask** - Author mode will always ask the user where he wants to open a DITA map file.

Schema Design

The **Schema Design** editor preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Pages** > **Schema Design**

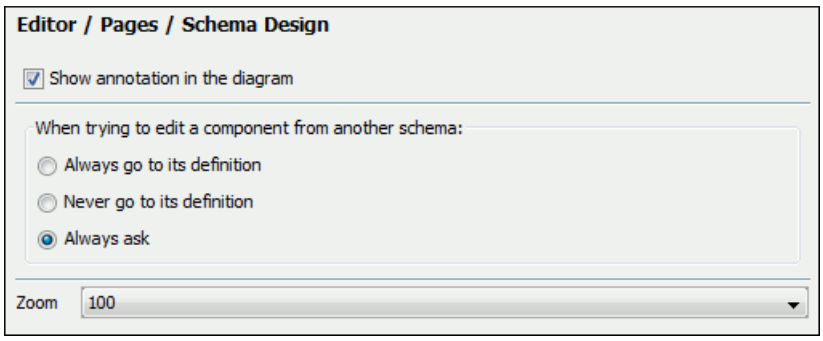


Figure 368: The Schema Design Preferences panel

The preferences for the XML Schema diagram are the following:

- **Show additional attributes in the diagram** - If checked the symbols for elements will include also properties like the name of the referred element (in case of a reference symbol), the base type, etc.
- **Show annotation in the diagram** - The content of `xs:documentation` elements is displayed only if this option is checked.
- **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** - The user is always asked where he wants to edit the component definition.
- **Zoom** - A zoom factor for the schema components. The range of possible values is from 25% to 300%. A custom zoom factor can be typed too.

Properties

You can decide to show additional properties for XML Schema components in the diagram and customize the properties to be displayed for each schema component.

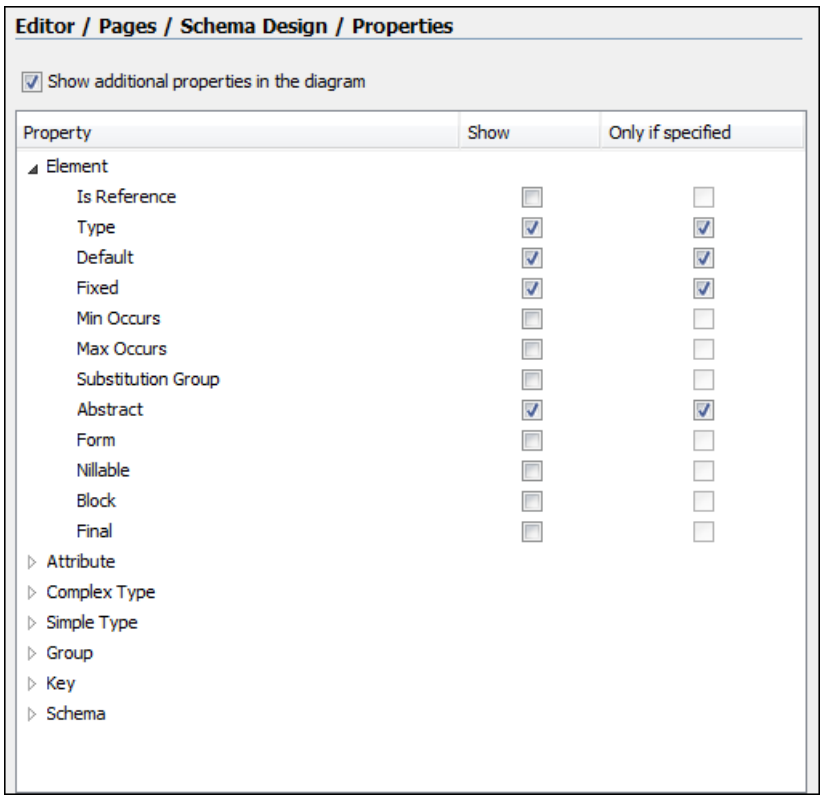


Figure 369: The Schema Properties Preferences Panel

For a component's properties you can decide if you want to display them only when having a specified value or all the time.

Format

The **Format** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Format** .

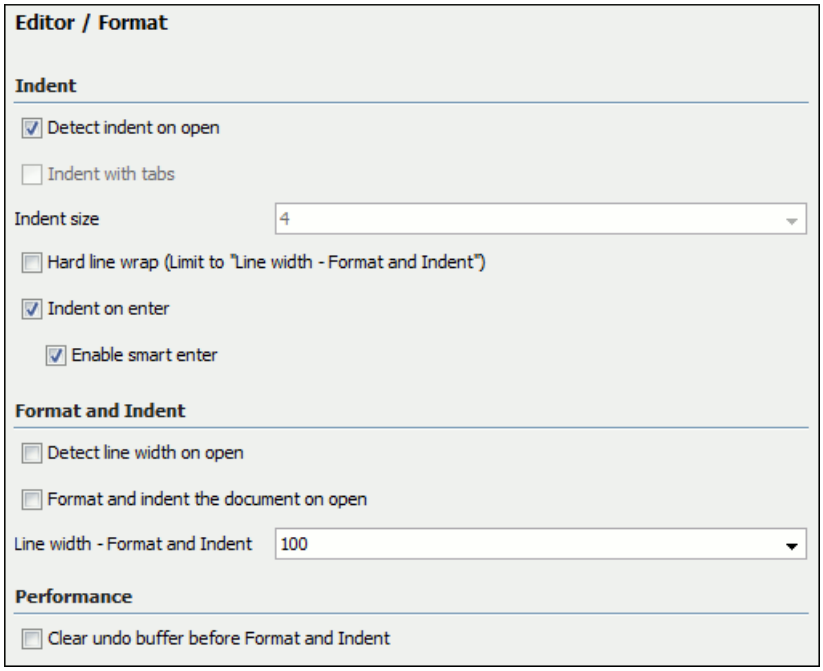


Figure 370: The Format Preferences Panel

The formatting preferences are the following:

- **Detect indent on open** - The editor tries to detect the indent settings of the opened XML document. In this way you can correctly format (pretty-print) files that were created with different settings, without changing your options. More than that you can activate the advanced option for detecting the maximum line width to be used for formatting and hard wrap. These features were designed to minimize the differences created by the pretty print operation when working with a versioning system, like CVS for example.
- **Indent with tabs** - When checked enables **Indent with tabs** to set the indent to a tab unit. When unchecked, the indent will measure as many spaces as needed in order to go to the next tab stop position. The maximum number of space characters is defined by the **Indent size** option.
- **Indent size** - Sets the number of spaces or the tab size that will equal a single indent. The indent can be spaces or a tab, selected by the preference **Indent with tabs**. For example if set to 4 one tab will equal 4 white spaces or 1 tab with size of 4 characters depending on the value of the **Indent with tabs** option.
- **Hard line wrap** - This feature saves time when writing an XML document with long lines. You can set a limit for the length of the lines in your document. When this limit is exceeded the editor will insert a new line before the word that breaks the limit, and indent the next line. This will minimize the need of reformatting the document.
- **Indent on Enter** - If checked, it indents the new line introduced when pressing Enter.
- **Enable Smart Enter** - If checked, it inserts a new indented line between start and end tag when Enter is pressed.
- **Detect line width on open** - If checked, it detects the line width automatically when the document is opened.
- **Format and indent the document on open** - When checked, an XML document will be formatted and indented before opening it in the editor panel. This option applies only to documents associated with the XML editor, not to documents associated with the XSD editor, RNG editor or XSL editor. This option does not apply on read-only documents when the *Can edit read only files* option is disabled.
- **Line width - Format and Indent** - Defines the point at which the **Format and Indent** (pretty-print) function will perform hard line wrapping. So if set to 100 pretty-print will wrap lines at the 100th space inclusive of white spaces, tags and elements.
- **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.

XML

The XML Format preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Format** > **XML** .

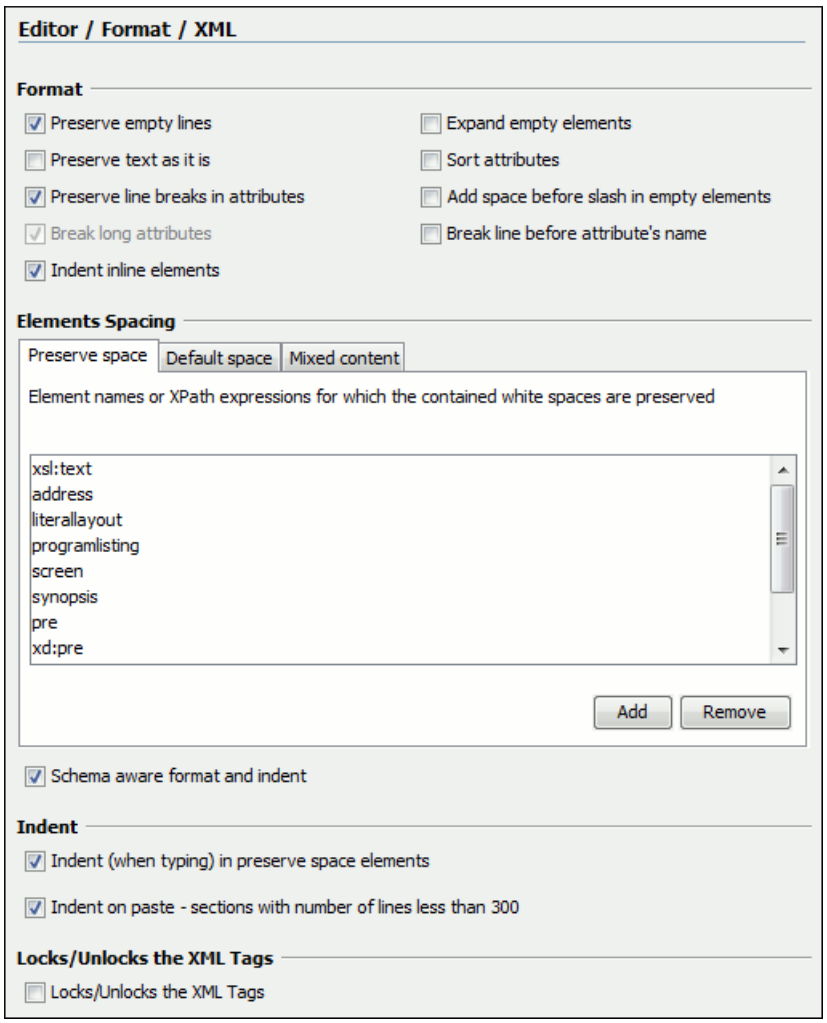


Figure 371: The XML Format Preferences Panel

The formatting preferences specific for XML files are the following:

- **Preserve empty lines** - When checked, the **Format and Indent** operation preserves all empty lines found in the document on which the pretty-print operation is applied.
- **Preserve text as it is** - If checked, the **Format and Indent** operation preserves text nodes as they are without removing or adding any whitespace.
- **Preserve line breaks in attributes** - If checked, the **Format and Indent** operation preserves the line breaks found in attributes. When this option is checked, **Break long lines** option is automatically disabled.
- **Break long attributes** - If checked, the **Format and Indent** operation breaks long attributes.
- **Indent inline elements** - If checked, the inline elements are broken and indented on separate lines if there are white spaces to the left and they follow another element start or end tag. Inline elements are elements which appear in a mixed-content context (parents with both non-whitespace text and elements). 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** - When checked, the **Format and Indent** operation outputs empty elements with a separate closing tag, ex. <a atr1="v1">. When not checked, the same operation represents an empty element in a more compact form: <a atr1="v1"/>.
- **Sort attributes** - When checked, the **Format and Indent** operation sorts the attributes of an element alphabetically. When not checked, the same operation leaves them in the same order as before applying the operation.
- **Add space before slash in empty elements** - Inserts a space character before the trailing / and > of empty elements.

  👉 **Note:** When formatting XHTML files, Oxygen always inserts a space character before the trailing / and > of empty elements.

- **Break line before attribute name** - If checked, the **Format and Indent** operation breaks the line before the attribute name.
- **Preserve space elements (XPath)** - This list contains simplified XPath expressions for the names of the elements for which the contained white spaces like blanks, tabs and newlines are preserved by the **Format and Indent** operation. The allowed XPath expressions are of the form:
  - `elementName`
  - `//elementName`
  - `/elementName1/elementName2/elementName3`
  - `//xs:localName`

  The namespace prefixes like `xs` in the previous example are treated as part of the element name without taking into account its binding to a namespace.
- **Default space elements (XPath)** - This list contains the names of the elements for which contiguous white spaces like blanks, tabs, and newlines are merged by the **Format and Indent** operation into one blank.
- **Mixed content elements (XPath)** - The elements from this list are treated as mixed when applying the **Format and Indent** operation, meaning that the operation breaks the line only when whitespaces are encountered.
- **Schema aware format and indent** - When checked, the **Format and Indent** operation takes into account the schema information regarding the space preserve, mixed, or element only property of an element.
- **Indent (when typing) in preserve space elements** - If checked, automatic tags indentation while editing takes place for all elements including the ones that are excluded from **Format and Indent** (default behavior). When unchecked, indentation while editing does not take place in elements that have the `xml:space` attribute set on `preserve` or are added to the list of **Preserve space elements**.
- **Indent on paste** - Indent paste text corresponding to the indent settings set by the user. This is useful for keeping the indent style of text copied from other document.
- **Locks / Unlocks the XML tags** - The default state of the opened editors. For more information see the *Locking and Unlocking XML markup* section.

Whitespaces

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 by the following actions:

- when the action **Format and Indent** 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, 10, 13 and 32 are always in the group of whitespace characters that must be normalized so they are always enabled in this panel.

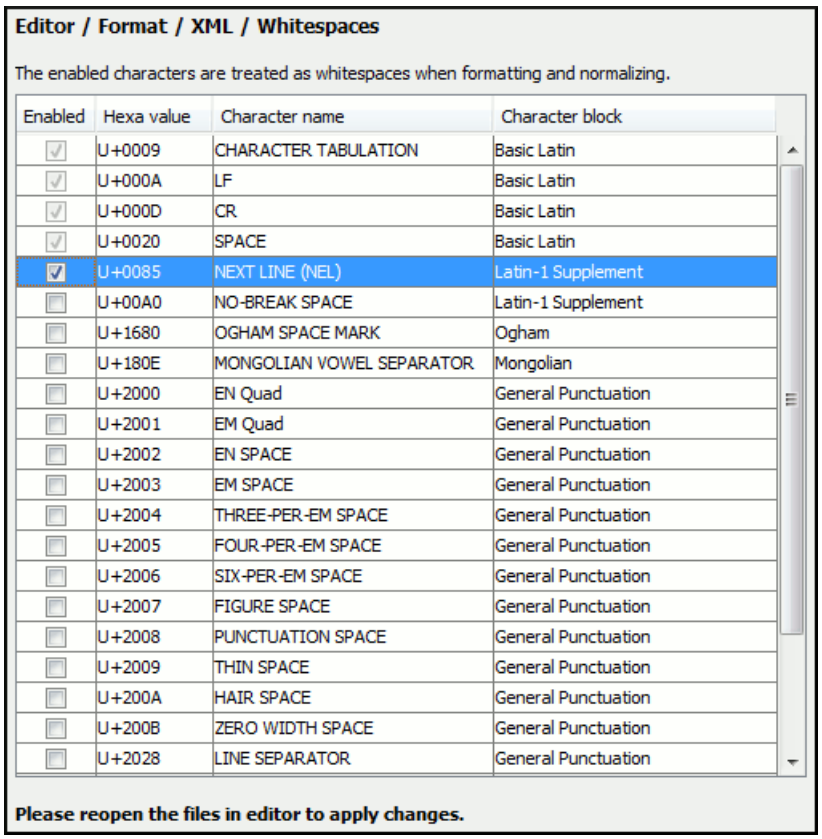


Figure 372: The Whitespaces Preferences Panel

CSS

The CSS Format preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Format** > **CSS** .

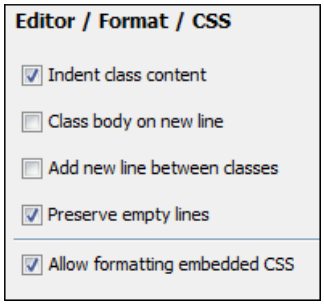


Figure 373: The CSS format preferences panel

The CSS formatting preferences are the following:

- **Indent class content** - If checked, the class content is indented during a **Format and Indent** operation. Enabled by default.
- **Class body on new line** - If checked, the class body (including the curly brackets) are placed on a new line after a **Format and Indent** operation.
- **Add new line between classes** - If checked, an empty line is added between two classes after a **Format and Indent** operation is performed.
- **Preserve empty lines** - If checked, the empty lines from the CSS content are preserved during the execution of a **Format and Indent** operation. Enabled by default.
- **Allow formatting embedded CSS** - If checked, the CSS content embedded in XML is formatted when the XML content is formatted by the **Format and Indent** operation. Enabled by default.

JavaScript

The JavaScript format preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Format** > **JavaScript** .

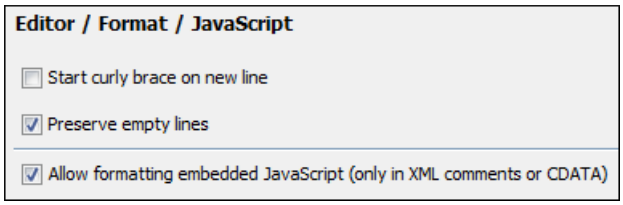


Figure 374: The JavaScript Format Preferences Panel

The JavaScript formatting preferences are the following:

- **Start curly brace on new line** - If true, opening curly braces will start on a new line. Otherwise they will remain on the same line as previous content of the JavaScript file.
- **Preserve empty lines** - If true, empty lines in the JavaScript code will be preserved.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows the application to format embedded JavaScript code, taking precedence over the *Schema aware format and indent* option.

Content Completion

The *content completion feature* enables inline syntax lookup and auto completion of mark-up elements and attributes to streamline mark-up and reduce errors while editing. These settings define the operating mode of the content assistant.

The **Content Completion** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Content Completion** .

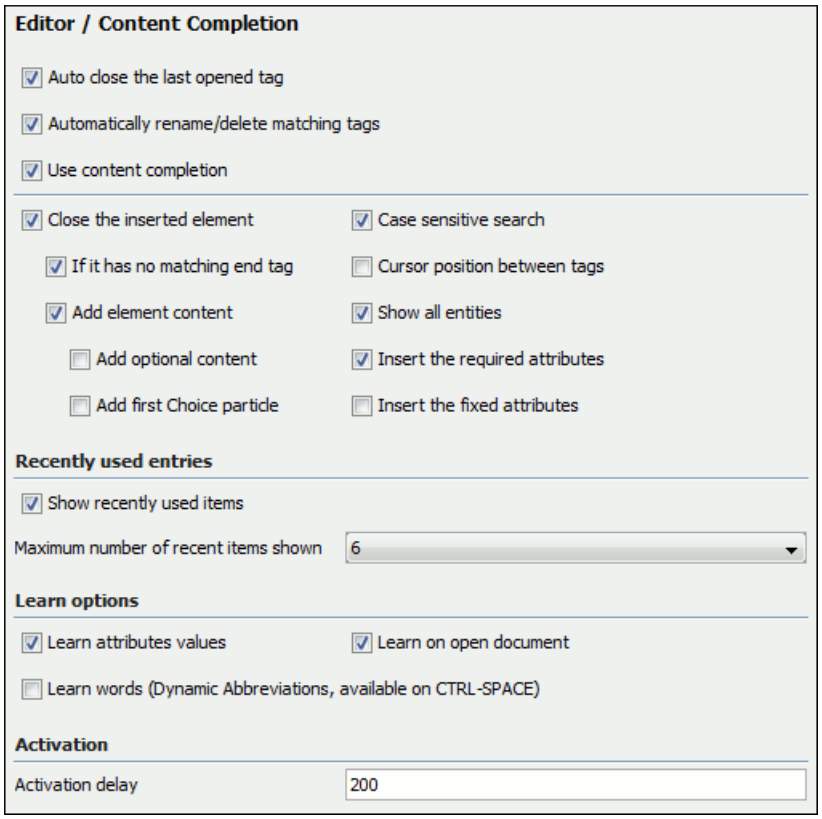


Figure 375: The Content Completion Preferences Panel

The configurable content completion preferences are the following:

- **Auto close the last opened tag** - If the **Use Content Completion** option is not checked but this option is checked, Oxygen will close the last opened tag when `</` is typed.
- **Automatically rename matching tag** - If checked, Oxygen will automatically rename the matching end tag when the start tag is modified in the editor.
- **Use Content Completion** - When unchecked, all content completion features are disabled.
- **Close the inserted element** - When inserting elements from the content completion assistant, both start and end tags are inserted.
- **If it has no matching tag** - When checked, the end tag of the inserted element will be automatically added only if it is not already present in the document.
- **Add element content** - When checked, Oxygen will insert automatically the required elements from the DTD or XML Schema or RELAX NG schema. This option is applied also in the Author mode of the XML editor.
- **Add optional content** - When checked, Oxygen will insert automatically the optional elements from the DTD or XML Schema or RELAX NG schema. This option is applied also in the Author mode of the XML editor.
- **Add first Choice particle** - When checked, Oxygen will insert automatically the first choice particle from the DTD or 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.
- **Case sensitive search** - When checked, the search in the content completion 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 checked, Oxygen will set the cursor automatically between tags. If the auto-inserted elements have attributes that are not required, the position of cursor can be forced between tags instead of inside the start tag.
- **Show all entities** - When checked, Oxygen will display 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** - When checked, Oxygen will insert automatically the required 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.
- **Insert the fixed attributes** - When checked, Oxygen will insert automatically 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 will remember the last inserted items from the content completion window. The number of items to be remembered is limited by the **Maximum number of recent items shown** combo box. These most frequently used items are displayed on the top of the content completion window and are separated from the rest of the suggestions by a thin grey line. This option is applied also in the Author mode of the XML editor.
- **Maximum number of recent items shown** - Limits the number of recently used items presented at the top of the content completion window. This option is applied also in the Author mode of the XML editor.
- **Learn attributes values** - When checked, Oxygen will display a list with all attributes values learned from the current document. This option is applied also in the Author mode of the XML editor.
- **Learn on open document** - When checked, Oxygen will automatically learn the document structure when the document is opened. This option is applied also in the Author mode of the XML editor.
- **Learn words (Dynamic Abbreviations, available on CTRL+SPACE)** - When checked, Oxygen will automatically learn the typed words and will make them available in a content completion fashion by pressing **(CTRL+SPACE)**.

  👉 **Note:** In order to be learned, the words need to be separated by space characters.

- **Activation delay of the proposals window (ms)** - Delay in milliseconds from last key press until the content completion window is displayed.

Annotations

The **Annotations** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Content Completion** > **Annotations** .

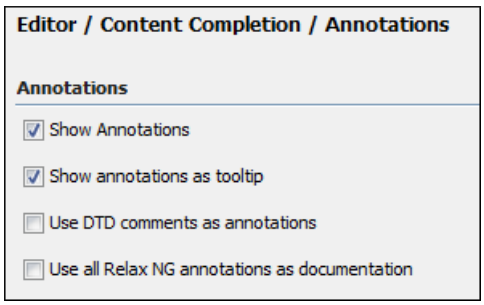


Figure 376: The Content Completion Annotations Preferences Panel

The following preferences can be configured for the annotations of the elements and attributes displayed by the content completion assistant:

- **Show annotations** - When checked, Oxygen will display the schema annotations that are present in the used schema for the current element, attribute or attribute value from the content completion window. This option is applied also in the Author mode of the XML editor.
- **Show annotations as tooltip** - When checked, it shows the annotation of an elements and attributes as a tooltip when the mouse pointer hovers over that element or attribute 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 checked, Oxygen will use all DTD comments as annotation. If it is not checked the following decision is performed: if among the gathered comments there are special Oxygen `doc:` comments, only those will be presented. If not, all encountered comments will be presented.
- **Use all Relax NG annotations as documentation** - When checked, any element that is not from the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0` will be considered annotation and will be displayed in the annotation window next to the content completion window and in the **Model** view. When unchecked only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` will be considered annotation.

XSL

The XSL preferences panel defines what elements are suggested by the content assistant of the XSL editor in addition to the XSL elements. It is opened from menu **Options** > **Preferences** > **Editor** > **Content Completion** > **XSL** .

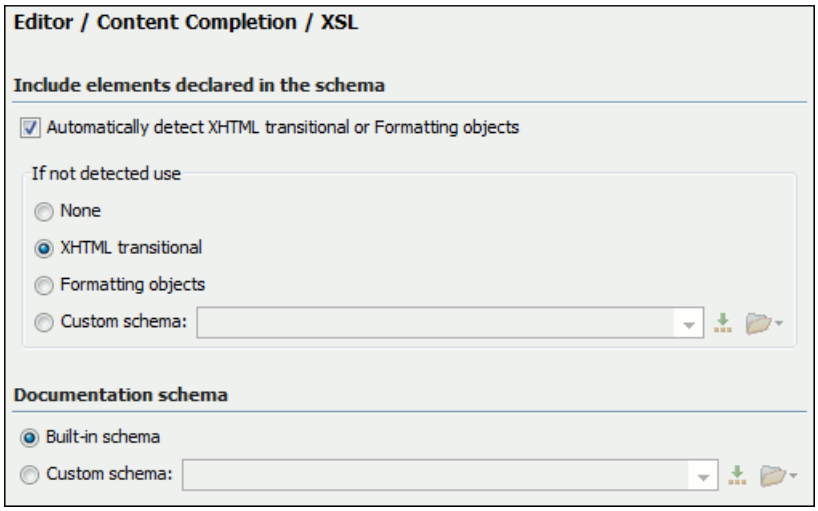


Figure 377: The Content Completion XSL Preferences Panel

You can choose to automatically detect if the XSL should use the XHTML or FO schemas for content completion based on the namespaces declared on the root element. If the detection fails, the following options will apply:

- **None** - The content completion will offer only the XSL elements.

- **XHTML transitional** - Includes XHTML Transitional elements as substitutes for `xsl:element`.
- **Formating objects** - Includes Formating Objects elements as substitutes for `xsl:element`.
- **Other** - 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 which will be used for documenting XSL stylesheets. Either select the built-in schema or choose a custom one. Supported schemas types are: XSD, RNG, RNC, DTD and NDVL.

XPath

The XPath preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Content Completion** > **XPath** .

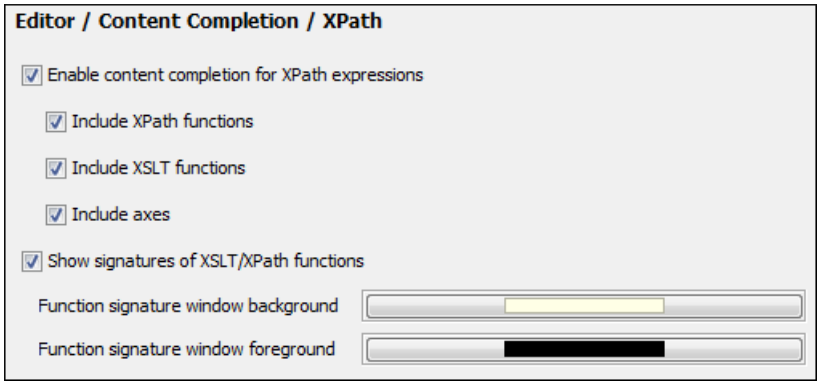


Figure 378: The Content Completion XPath Preferences Panel

The preferences that allow configuring the content completion in the XPath expressions are the following:

- **Enable content completion for XPath expressions** - Disables and enables *content completion in XPath expressions* entered in the XSL attributes `match`, `select` and `test` 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 when editing XPath expressions.
- **Show signatures of XSLT / XPath functions** - If checked, the editor will indicate in a tooltip helper the signature of the XPath function located at the caret position. See the *XPath Tooltip Helper* section for more information.
- **Function signature window background** - The background color of the tooltip window.
- **Function signature window foreground** - The foreground color of the tooltip window.

XSD

These preferences define what elements are suggested by the content assistant, in addition to the ones from the XML Schema schema, inside the `xs:annotation/xs:appinfo` elements of an XML Schema.

The XSD preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Content Completion** > **XSD** .

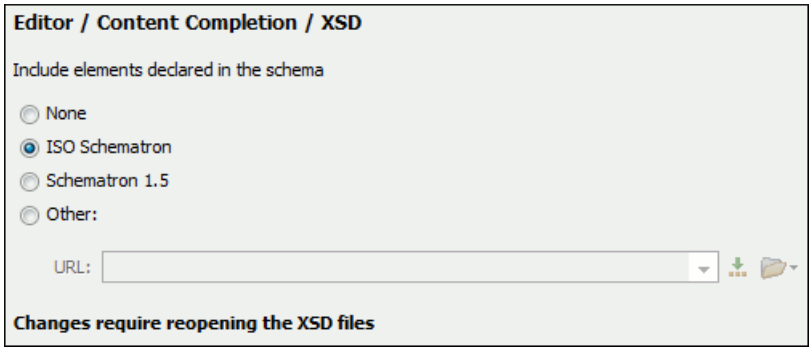


Figure 379: The Content Completion XSD Preferences Panel

The following preferences are available for the content completion in XML Schema files:

- **None** - The content completion will offer only the XML Schema schema information.
- **ISO Schematron** - Includes ISO Schematron elements in `xs:appinfo`.
- **Schematron 1.5** - Includes Schematron 1.5 elements in `xs:appinfo`.
- **Other** - Includes in `xs:appinfo` elements from an XML Schema specified from a URL.

Colors

Oxygen supports syntax highlight for XML, DTD, Relax NG (XML and Compact Syntax), Java, JavaScript, PHP,CSS, XQuery, C++, C, Perl, Properties, SQL, Shell and Batch documents. While Oxygen provides a default color configuration for highlighting the tokens, you may choose to customize it, as required, using the Colors preferences panel.

The Colors preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Colors** > **Syntax Highlight** .

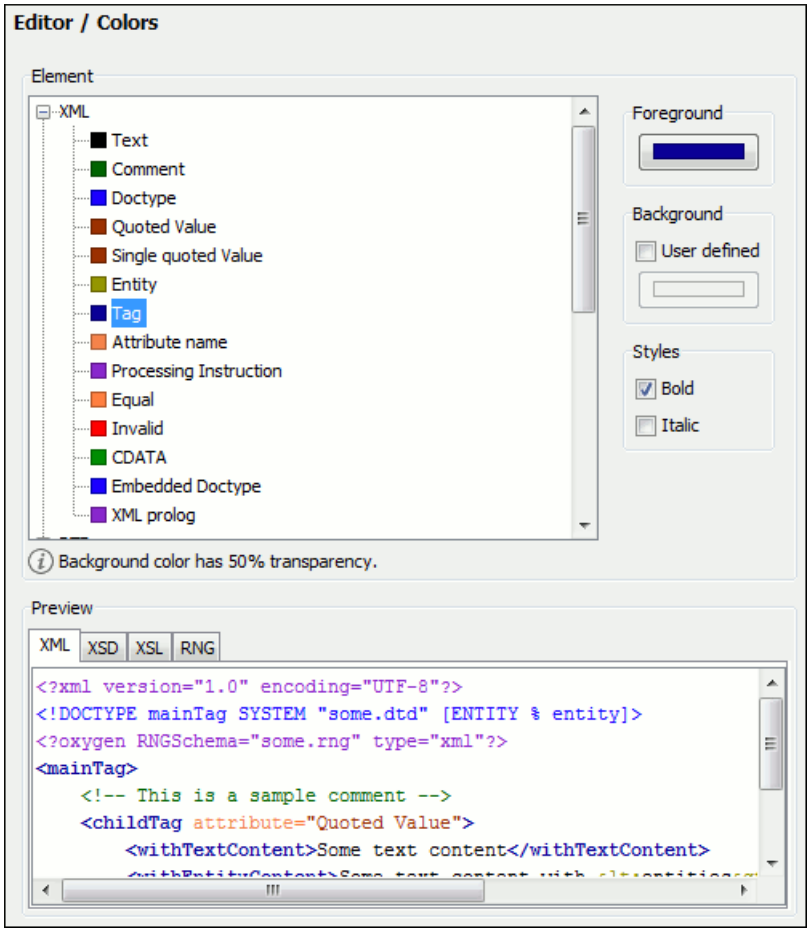


Figure 380: The Colors Preferences Panel

For each document type there is a set of tokens. When a document type node is expanded, the associated tokens are listed. For each token the color and the font style that is configured here will be used in Text mode of the editor panel. The tokens for XML documents are used also in XSD, XSL, RNG documents so the **Preview** area has 4 tabs when an XML token is selected in the **Element** area : XML, XSD, XSL, RNG.

When you don't know the name of the token that you want to configure just 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 color** - The **Foreground** button opens a color dialog that allow setting the color properties for the selected token with one of the methods: Swatches, HSB or RGB.

  - **Swatches** - Displays a color palette containing a variety of colors from across the color spectrum and shades thereof. Select a color.
  - **HSB** - Hue, Saturation and Brightness (HSB) enables you to specify a color by describing it using hue, saturation and brightness.
  - **RGB** - Red, Green and Blue (RGB) enables you to specify a color using triplets of red, green and blue numbers.
- **Background color** - The **Background** button opens the same color dialog as the **Foreground** button.
- **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 **Preview** panel displays the appearance of all token colors in a sample document as they will be rendered in the editor.

Modifications are saved when the **OK** button is clicked. The **Restore Defaults** button changes all the token colors to the default values.

Elements / Attributes by Prefix

The **Elements / Attributes by Prefix** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Colors** > **Elements / Attributes by Prefix** .

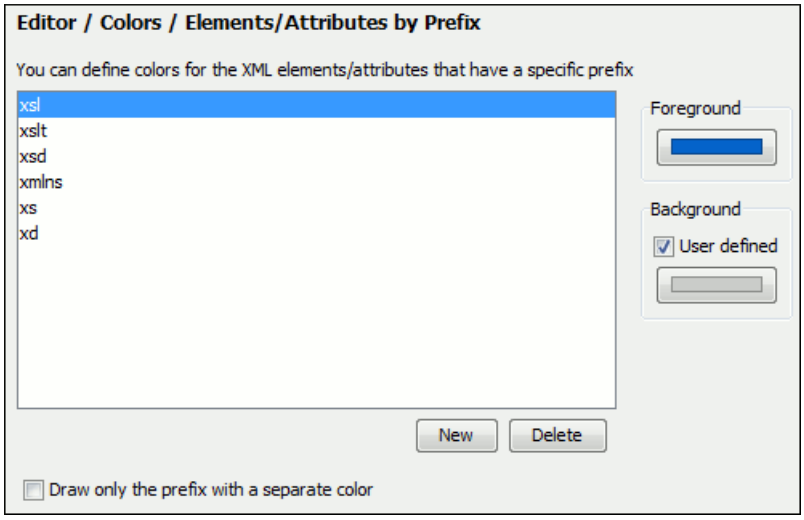


Figure 381: The Elements / Attributes by Prefix Preferences Panel

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 the prefix will be marked with the same color.

You can edit the following color properties of the selected token:

- **Foreground color** - The **Foreground** button opens a color dialog that allow setting the color properties for the selected token with one of the methods: Swatches, HSB or RGB.
  - **Swatches** - Displays a color palette containing a variety of colors from across the color spectrum and shades thereof. Select a color.
  - **HSB** - Hue, Saturation and Brightness (HSB) enables you to specify a color by describing it using hue, saturation and brightness.
  - **RGB** - Red, Green and Blue (RGB) enables you to specify a color using triplets of red, green and blue numbers.
- **Background color** - 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 checking the **Draw only the prefix with a separate color** option.

Open / Save

The **Open / Save** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Open / Save** .

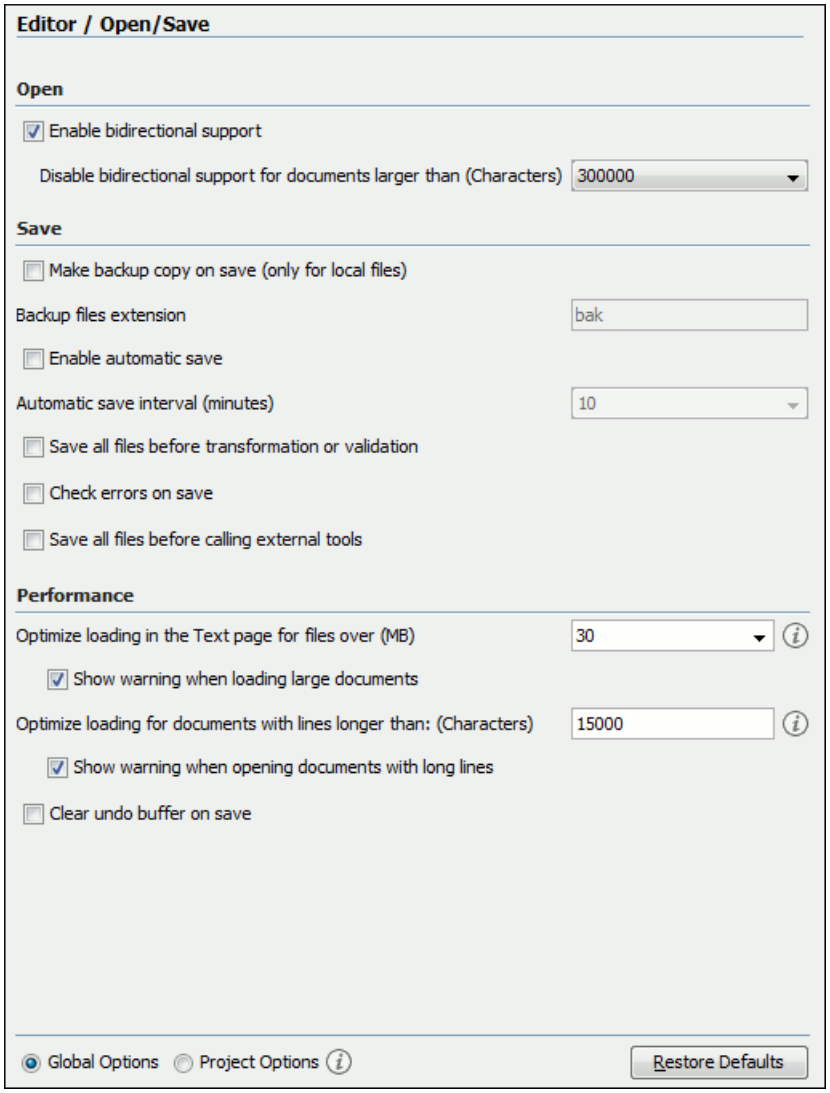


Figure 382: The Open / Save Preferences Panel

The preferences related with opening and saving documents are the following:

- **Format document when longest line exceeds** - Specifies the default behavior when the longest line of a document exceeds the specified limit. This option does not apply on read-only documents when the ***Can edit read only files*** option is disabled. You can choose between:
  - **Always format** - Runs the action **Format and Indent** on a document with very long lines when opening it without notification for the user.
  - **Never format** - Never modifies a document with very long lines on opening it.
  - **Always ask** - Asks the user if he wants to run the action **Format and Indent** on every open of a document with very long lines.
- **Characters limit for bidirectional text documents** - Specifies the characters limit for bidirectional text documents. If the total number of characters found in a document exceeds this limit, the bidirectional support is disabled.

- **Make backup copy on save (only for local files)** - If checked, a backup copy is made when saving the edited document. The default extension is `.bak`, but you can change extension as you prefer. The option of making a backup copy on save is available only for local files (files that are stored on the local file system).
- **Enable automatic save** - Automatic save is a useful feature that ensures your work is being saved in the background. You can specify the time intervals between automatic saves.
- **Automatic save interval (minutes)** - Selects the interval in minutes between two automatic save actions.
- **Save all files before transformation or validation** - Saves all opened files before validating or transforming an XML document. In this way the dependencies are resolved, for example when modifying both the XML document and its XML Schema.
- **Check errors on save** - If checked, a checking for errors is done when saving the edited document.
- **Save all files before calling external tools** - If checked, all files are saved before executing an external tool.
- **Optimize loading in the Text page for files over (MB)** - File loading is optimized for reduced memory usage for any file with a size larger than this value. This optimization is useful for being able to load and edit very large files (hundreds of MB) and it comes with *a few restrictions* on memory-intensive operations.
- **Show warning when loading large documents** - A warning dialog is displayed when you try to load a very large file.
- **Optimize loading for documents with lines longer than (Characters)** - Line wrap is turned on for a document containing lines that exceed the length specified with this option. For a list of the restrictions applied to a document with long lines please see *the **Editing Documents with Long Lines** section.*
- **Show warning when loading documents with long lines** - If checked, a warning dialog is displayed when you try to open a file that contains at least one line that exceeds the maximum line length specified in the previous option. The warning dialog informs you that line wrapping is turned on and some of the editing features are disabled. Another option is to *format and indent the document* after it is opened in the editor panel. For a list of the restrictions applied to a document with long lines please see the section about *formatting documents with long lines*.
- **Clear undo buffer on save** - If checked, you cannot undo anymore editing actions that preceded the save operation. 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 performing modifications on very large documents.
- **Consider application bundles to be directories when browsing** - This option is available only on the Mac OS X platform. When checked the file browser dialog allows browsing inside an application bundle as in a regular folder. When unchecked the file browser dialog does not allow browsing inside an application bundle, as the Finder application does on Mac OS X. The same effect can be obtained by setting the property *apple.awt.use-file-dialog-packages* to true or false in the `Info.plist` descriptor file of the Oxygen application by adding two lines in this descriptor file:

```
<key>apple.awt.use-file-dialog-packages</key>
<string>false</string>
```

Templates

This panel groups the preferences that are related with code templates and document templates:

- *Code Templates*
- *Document Templates*

Code Templates

Code templates are small document fragments that can be inserted quickly at the editing position and can be reused in other editing sessions. Oxygen 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, you use:

- The shortcut key for content completion on request: **(Ctrl+Space)** on Windows and Linux, **(Cmd+Space)** on Mac OS X. It displays the code templates in *the same content completion window with elements from the schema of the document*.

- The shortcut key for code templates on request: **(Ctrl+Shift+Space)** on Windows and Linux, **(Cmd+Shift+Space)** on Mac OS X. It displays only the code templates in the popup window.

The **Code Templates** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Templates** > **Code Templates** . It 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.

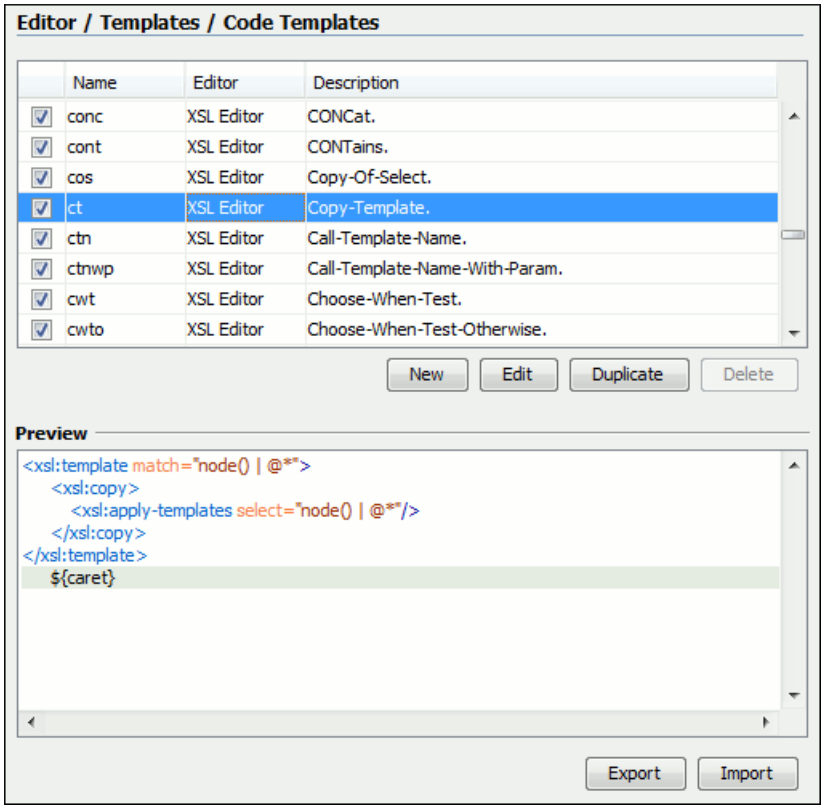


Figure 383: The Code Templates Preferences Panel

- **New** - Defines a new code template. You can define a code template for a specific type of editor or for all editor types.
- **Edit** - Edits the selected code template.
- **Duplicate** - Duplicates the code template that is selected in the list.
- **Delete** - Deletes the code template that is selected in the list. 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

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. You have to add the template files in a folder that is specified in this panel or in the `templates` folder of the Oxygen install directory.

The **Document Templates** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Templates** > **Document Templates** .

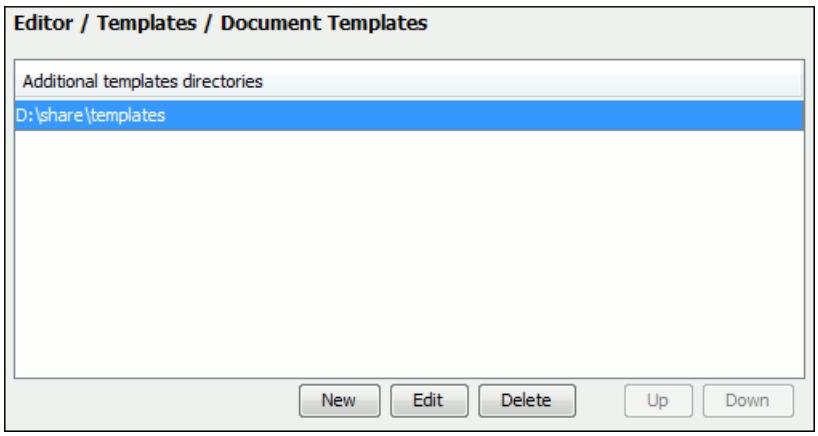


Figure 384: Document Templates Preferences Panel

A new template folder is added with the **New** button which opens the following dialog:

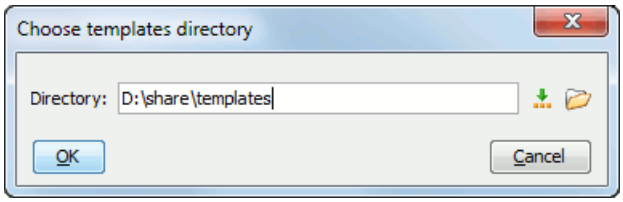


Figure 385: Document Templates Input Dialog

Spell Check

The **Spell Check** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Spell Check** .

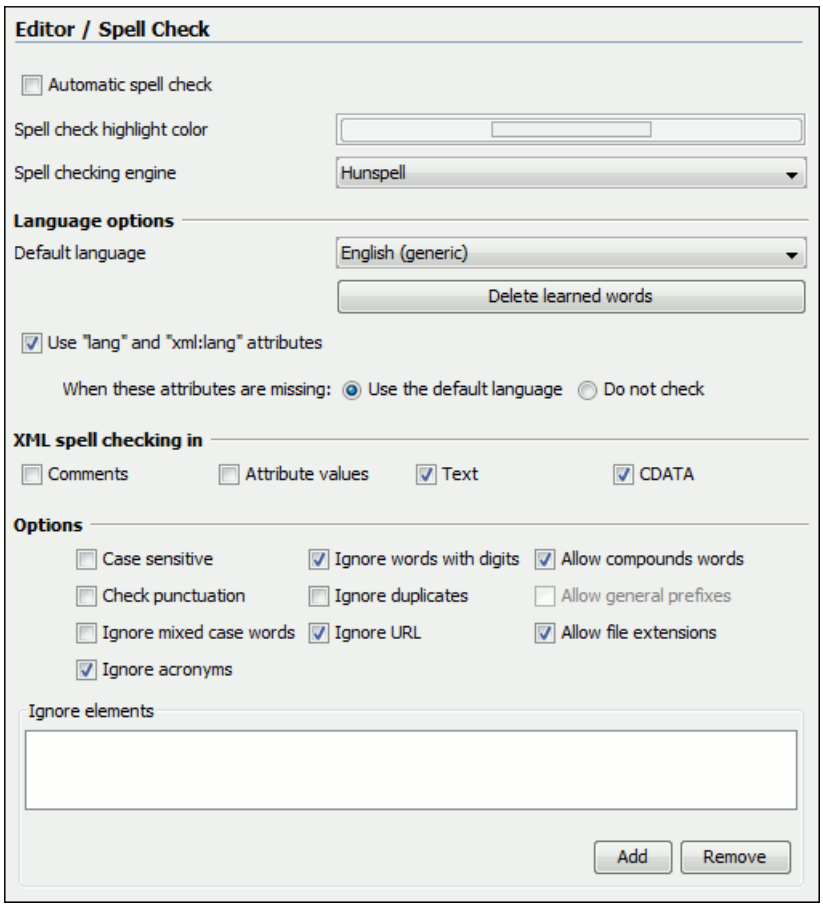


Figure 386: The Spell Check Preferences Panel

The spell check preferences are the following:

- **Automatic Spell Check** - When checked, the spell checker highlights the errors as you modify the document.
- **Spell check highlight color** - Use this option to set the color for highlighting the spell check errors.
- **Spell checking engine** - The engines available are Hunspell and AZ Check. 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** combo box.
- **Default language** - The default language combo allows you to choose the language used by default. If the language of your documents is not listed in this combo box you can *add a spelling dictionary* for your language which will be added to this list.
- **Delete learned words** - Press this button to open the list of learned words. Here you can select the items you want to remove.
- **Use "lang" and "xml:lang" attributes** - If checked, the contents of any element with such an attribute will be checked using a dictionary 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 the user to specify if the spell checker will be enabled inside XML comments, attribute values, text and CDATA sections.
- **Case sensitive** - When checked, spell checking reports capitalization errors, for example a word that starts with lowercase after *etc.* or *i.e.*.
- **Ignore mixed case words** - When checked, operations do not check words containing case mixing (e.g. *SpellChecker*).
- **Ignore words with digits** - When checked, the spell checker does not check words containing digits (e.g. *b2b*).
- **Ignore Duplicates** - When checked, the spell checker does not signal two successive identical words as an error.

- **Ignore URL** - When checked, ignores words looking like URL or file names (e.g. *www.oxygenxml.com* or *c:\boot.ini*) .
- **Check punctuation** - When checked, punctuation checking is enabled: misplaced white space and wrong sequences, like a dot following a comma, are highlighted as errors.
- **Allow compounds words** - When checked, all words formed by concatenating two legal words with an hyphen are accepted. If the language allows it, two words concatenated without hyphen are also accepted.
- **Allow general prefixes** - When checked, a word formed by concatenating a registered prefix and a legal word is accepted. For example if *mini-* is a registered prefix, the checker accepts *mini-computer*.
- **Allow file extensions** - When checked, accepts any word ending with registered file extensions (e.g. *myfile.txt*, *index.html*, etc.).
- **Ignore acronyms** - When checked, the acronyms are not reported as errors.
- **Ignore elements** - A list of XPath expressions for the elements that will be ignored by spell checking. Only a small subset of XPath expressions are supported, that is:
  - only the '/' and '//' separators
  - the '*' wildcard

  An example of allowed XPath expression: */a/*/b*.

Document Checking

The **Document Checking** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Document Checking** . It contains preferences for configuring how a document is checker for errors, that is well-formed errors and validation errors.

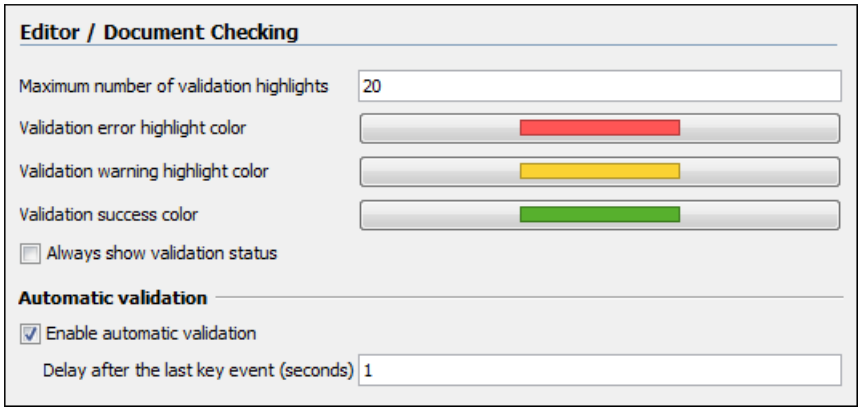


Figure 387: Document Checking Preferences Panel

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*.
- **Validation error highlight color** - The color used to highlight validation errors in the document.
- **Validation warning highlight color** - The color used to highlight validation warnings in the document.
- **Validation success color** - The color used to highlight in the vertical ruler bar the success of the validation operation.
- **Always show validation status** - If this option is selected the line at the bottom of the editor panel which presents the current validation error or warning is always visible. This is useful to avoid scrolling problems when **Automatic validation** is enabled and the vertical scroll bar may change position due to displaying an error message while the document is edited.
- **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen.
- **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

Mark Occurrences

The **Mark Occurrences** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Mark Occurrences** :

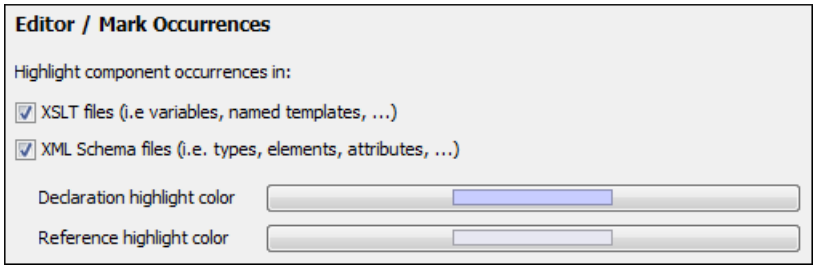


Figure 388: Mark Occurrences Preferences Page

The following preferences can be set:

- **XSLT files** - activates the *Highlight Component Occurrences* in XSLT files;
- **XML Schema files** - activates the *Highlight Component Occurrences* in XSD files;
- **Declaration highlight color** - color used to highlight the component declaration;
- **Reference highlight color** - color used to highlight component references.

Custom Validation Engines

The **Custom Validation Engines** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Custom Validations** .

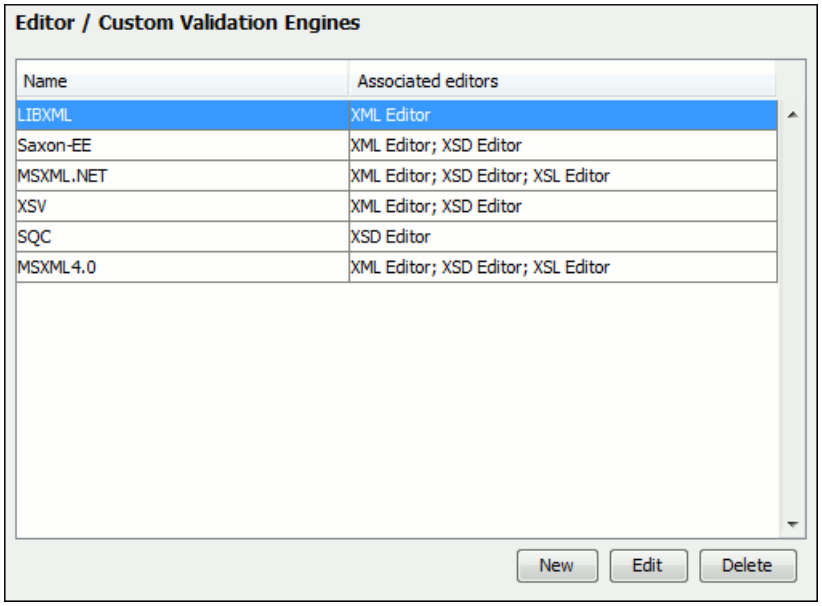


Figure 389: Custom Validation Preferences Panel

If you want to add a new custom validation tool or edit the properties of an exiting one you can use the **Custom Validator** dialog displayed by pressing the **New** button or the **Edit** button.

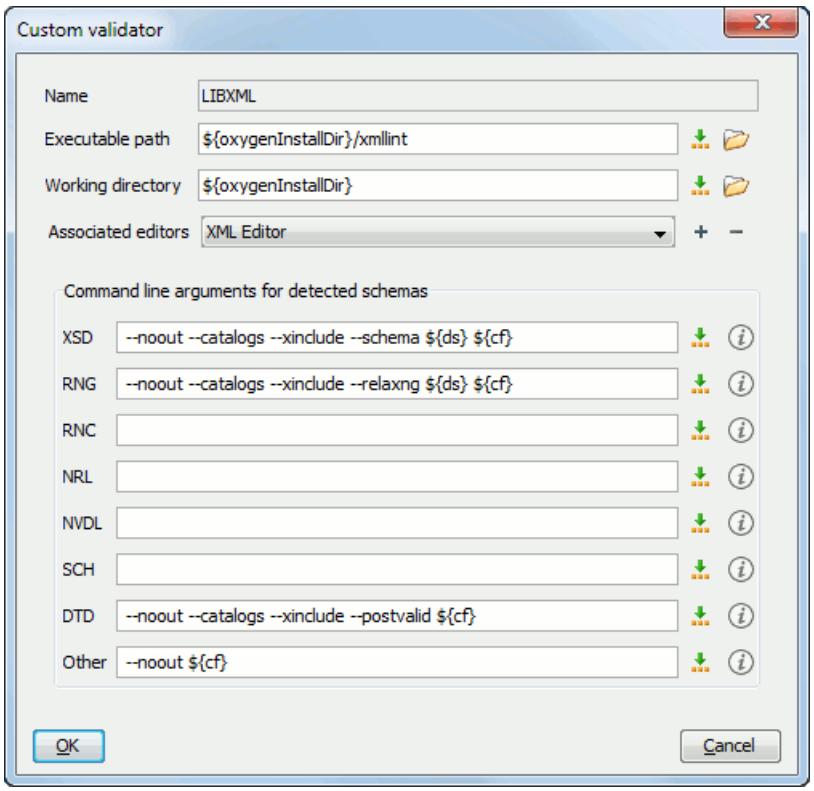


Figure 390: Edit a Custom Validator

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}*, *${oxygenInstallDir}*, etc.
- **Working directory** - The working directory of the custom validation tool. The following editor variables can be used in this field:
  - **${home}** - The path to the user home directory.
  - **${pd}** - The current project directory.
  - **${oxygenInstallDir}** - The Oxygen installation directory.
- **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:
  - **${cf}** - The path of the current file as a local file path.
  - **${cfu}** - The path of the current file as a URL.
  - **${ds}** - The path of the detected schema as a local file path.
  - **${dsu}** - The path of the detected schema as a URL.

CSS Validator

The **CSS Validator** preferences panel is opened from menu **Options** > **Preferences** > **CSS Validator** .

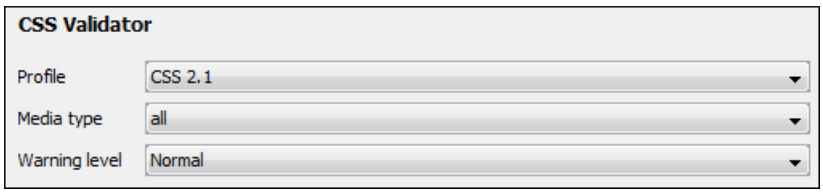


Figure 391: The CSS Validator Preferences Panel

The following options can be configured for Oxygen's built-in CSS validator:

- **Profile** - Selects one of the available validation profiles: CSS 1, CSS 2, CSS 2.1, CSS 3, SVG, SVG Basic, SVG Tiny, Mobile, TV Profile, ATSC TV Profile.
- **Media Type** - Selects one of the available mediums: all, aural, braille, embossed, hand-held, print, projection, screen.
- **Warning Level** - Sets the minimum severity level for reported validation warnings. It is one of: all, normal, most important, no warnings.

XML

This section describes the panels that contain the user preferences related with XML.

XML Catalog

The **XML Catalog** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XML Catalog** .

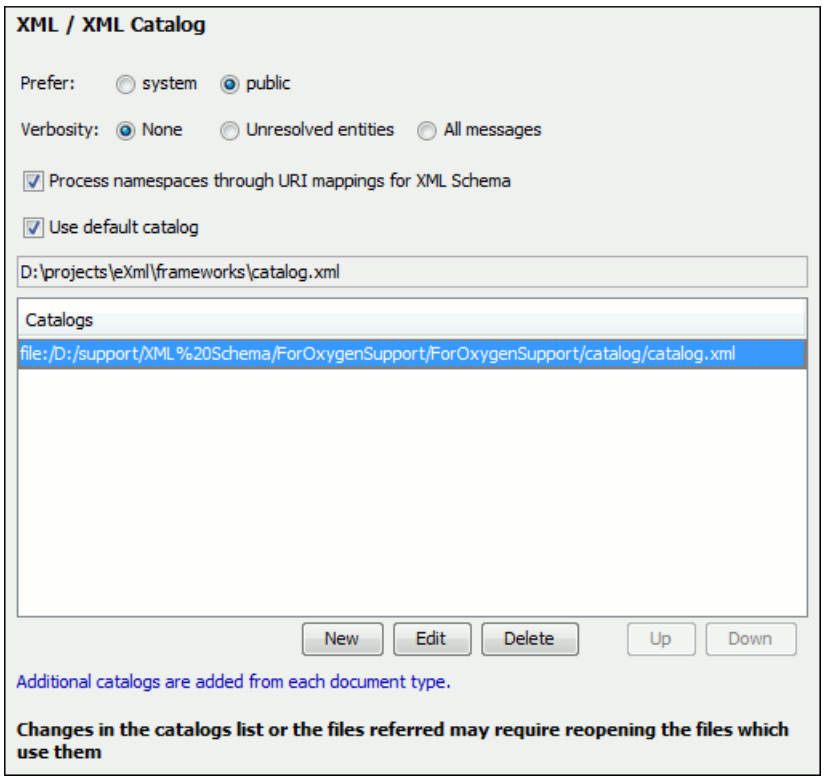


Figure 392: The XML Catalog Preferences Panel

The **Prefer** option is used to specify if Oxygen will try 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 the **Process namespaces through URI mappings for XML Schema** option is not selected only the schema location of an XML Schema that is declared in an XML document is searched in XML catalogs. If the option is selected the schema location of an XML Schema is searched and if it is not resolved the namespace of the schema is also searched.

If the **Use default catalog** option is checked the first XML catalog which Oxygen 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 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 you must reopen the current edited files which use the modified catalog or run *the action **Reset Cache and Validate*** so that the XML catalog changes take full effect.

XML Parser

The **XML Parser** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XML Parser** .

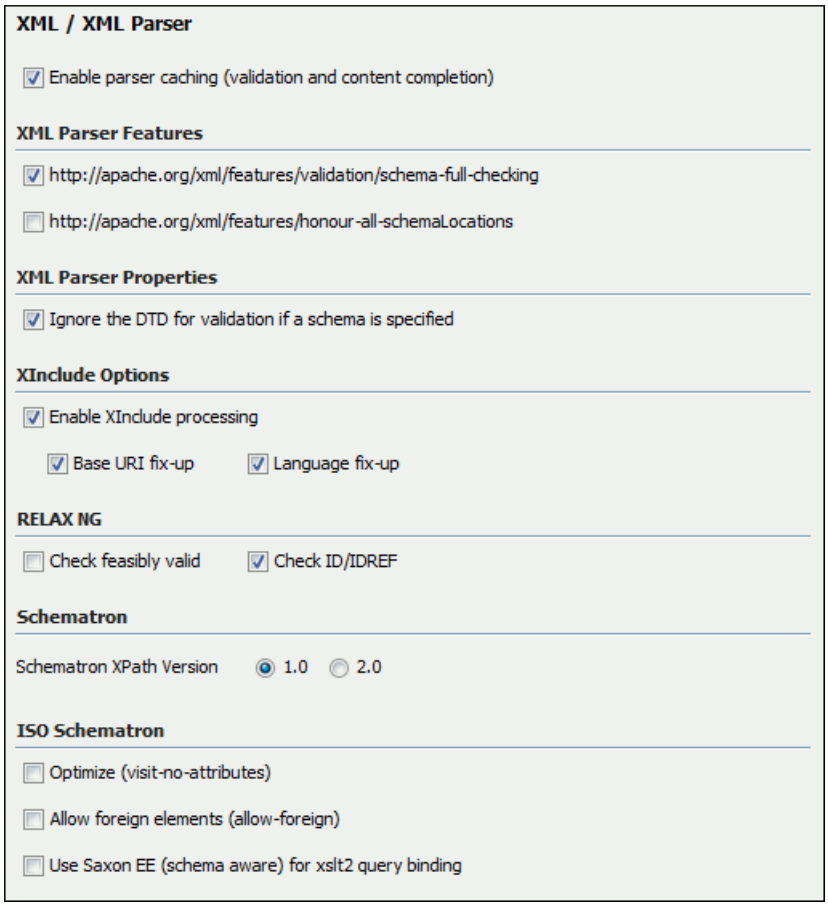


Figure 393: The XML Parser preferences panel

The configurable options of the built-in XML parser are the following:

- **http://apache.org/xml/features/validation/schema-full-checking** - Sets the *schema-full-checking* feature to true, that is a validation of the parsed XML document is performed against a schema (W3C XML Schema or DTD) while the document is parsed.
- **http://apache.org/xml/features/honour-all-schema-location** - Sets the *honour-all-schema-location* feature to true. This means all the files that declare W3C XML Schema components from the same namespace are used to compose the validation model. If this option is not selected only the first W3C XML Schema file that is encountered in the XML Schema import tree is taken into account.
- **Ignore the DTD for validation if a schema is specified** - Forces validation against a referred schema (W3C XML Schema, Relax NG schema, Schematron 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, a Relax NG schema or a Schematron schema.
- **Enable XInclude processing** - Enables XInclude processing. If checked, the XInclude support in Oxygen 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.
- **Check ID/IDREF** - Checks the ID/IDREF matches when the Relax NG document is validated.
- **Check feasibly valid** - Checks the Relax NG to be feasibly valid when this document is validated.
- **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)** - If 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** - If checked, Saxon EE will be used for `xslt2` query binding. If not checked, Saxon PE will be used instead.

Saxon EE Validation

The **Saxon EE Validation** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XML Parser** > **Saxon EE Validation** .

The following options are available:

- **XML Schema version** - allows you to select the version of W3C XML Schema for validation against XML Schema performed by the Saxon EE engine: XML Schema 1.0 or XML Schema 2.0.
- **XML Schema validation** - you can set Oxygen to use Saxon EE as default XML Schema validator. If enabled it is used to validate XML Schema and XML documents against an XML Schema. By default this option is turned off.

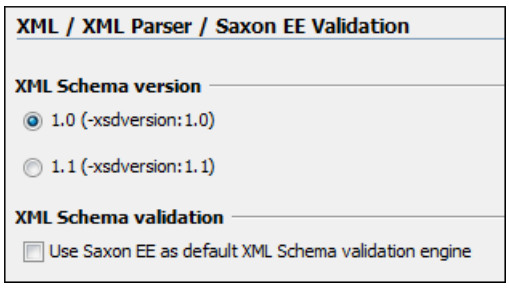


Figure 394: The Saxon EE Validation Preferences Panel

XML Instances Generator

The **XML Instances Generator** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XML Instances Generator** . It sets the default parameters of the **Generate Sample XML Files** tool that is available on the **Tools** menu.

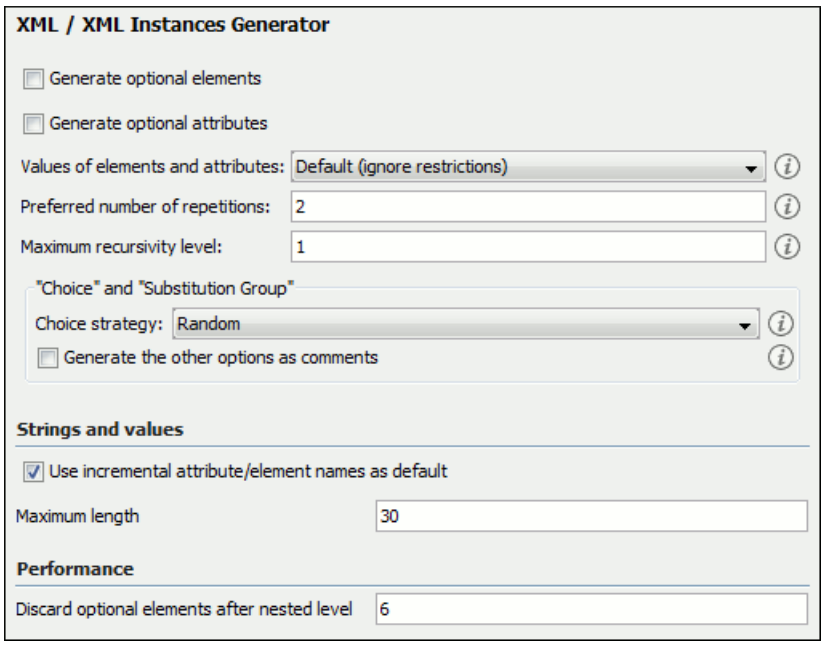


Figure 395: The XML Instances Generator Preferences Panel

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** - The number of repetitions for an element that has a big value of the `maxOccurs` attribute.
- **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.

- **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 will be: `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 huge XML documents.

XProc Engines

Oxygen comes with a built-in XProc engine called *Calabash*. An external XProc engine can be configured in this panel.

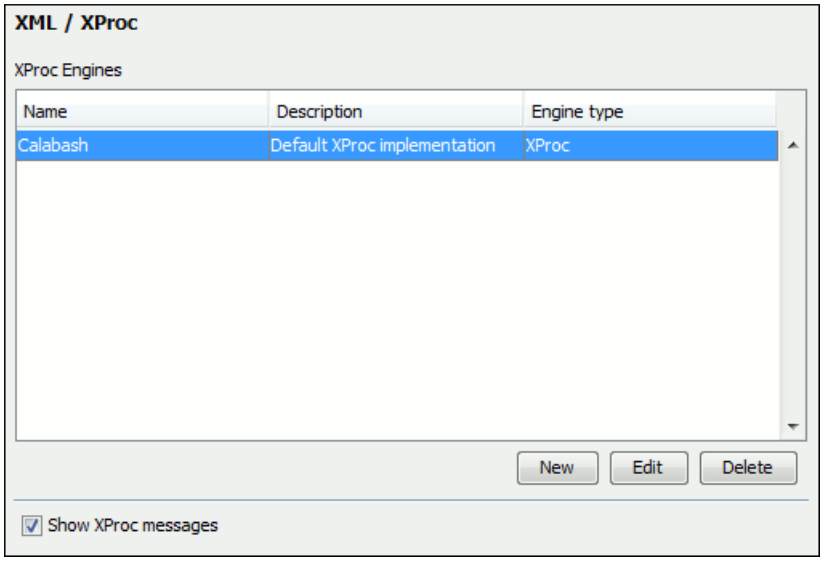


Figure 396: The XProc Preferences Panel

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.

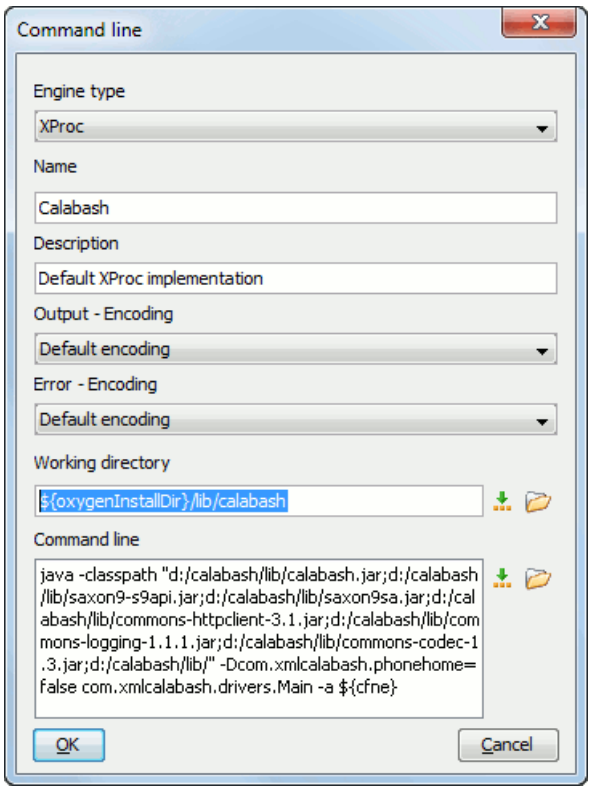


Figure 397: Creating an XProc external engine

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.

XSLT/FO/XQuery

The **XSLT/FO/XQuery** preferences panel is opened from menu **Options** > **Preferences** > **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.

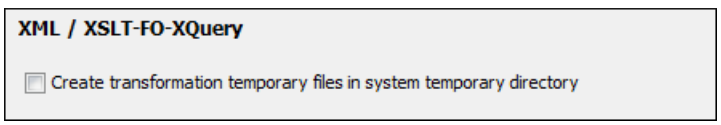


Figure 398: The XSLT/FO/XQuery Preferences Panel

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

The **XSLT** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** .

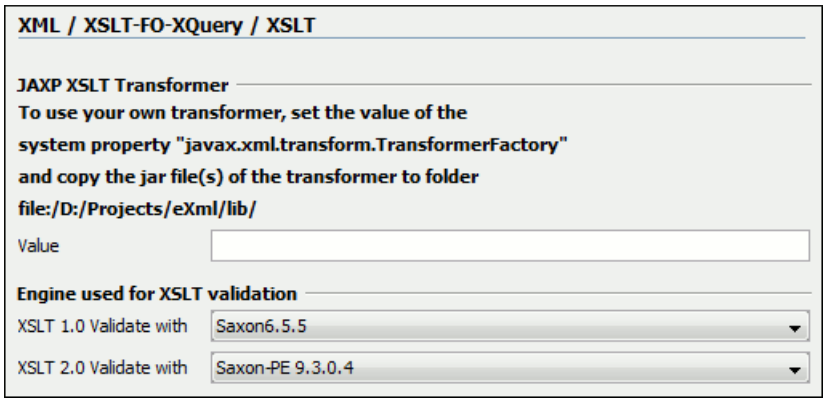


Figure 399: The XSLT Preferences Panel

If you want to use an XSLT transformer implemented in Java different than the ones that ship with Oxygen namely Apache Xalan and Saxon all you have to do is to specify the name of the transformer's factory class which Oxygen will set as the value of the Java property `javax.xml.transform.TransformerFactory`. For instance, to perform an XSLT transformation with Saxon 9.3.0.5 you have to place the Saxon 9.3.0.5 jar file in the Oxygen libraries folder (the `lib` subfolder of the Oxygen installation folder), set `net.sf.saxon.TransformerFactoryImpl` as the property value and select JAXP as the XSLT processor in the transformation scenario associated to the transformed XML document.

The XSLT preferences are the following:

- **Value** - Allows the user to enter the name of the transformer factory Java class.
- **XSLT 1.0 Validate with** - Allows the user to set the XSLT engine used for validation of XSLT 1.0 documents.
- **XSLT 2.0 Validate with** - Allows the user to set the XSLT Engine used for validation of XSLT 2.0 documents.

Saxon6

The **Saxon 6** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon 6** .

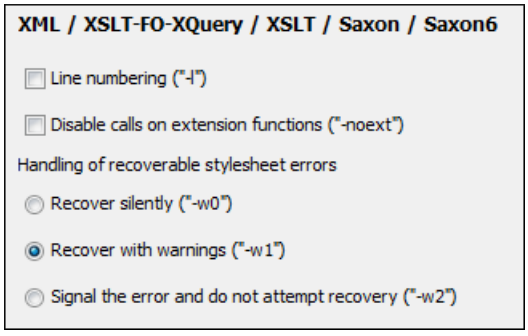


Figure 400: The Saxon 6 XSLT Preferences Panel

The built-in Saxon 6 XSLT processor can be configured with the following options:

- **Line numbering** - If checked, line numbers are maintained and reported in error messages for the XML source document.
- **Disable calls on extension functions** - If checked, 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 will be 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

The **Saxon HE/PE/EE** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon HE/PE/EE** .

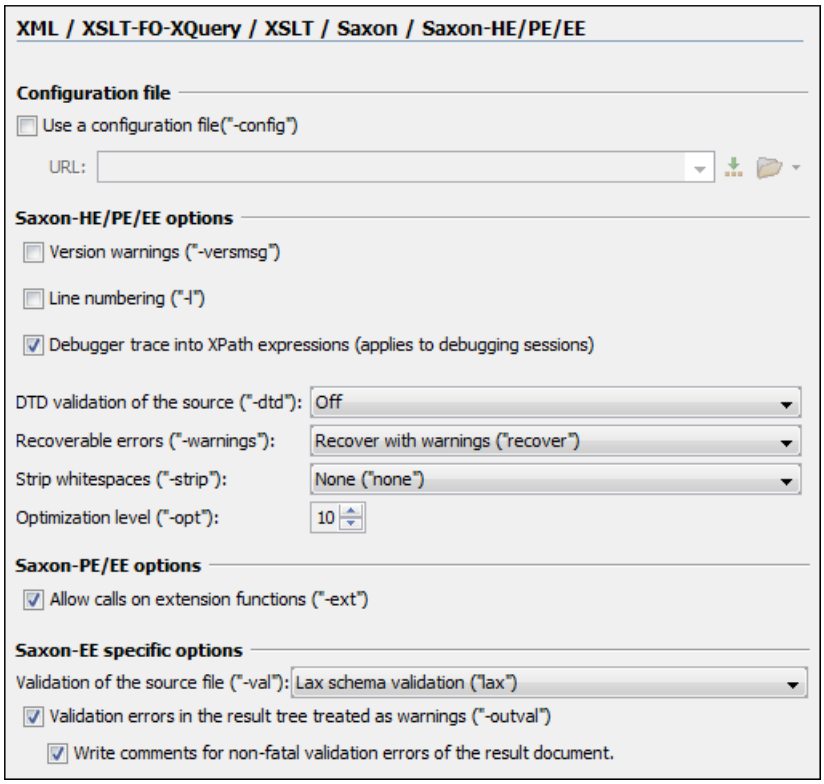


Figure 401: The Saxon HE/PE/EE XSLT preferences panel

The XSLT options which can be configured for the Saxon 9.3.0.5 transformer (all three editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that will be used for XSLT transformation and validation.
- **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.
- **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 error as non-fatal if it fails

  Note that 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. Either 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. Strips no whitespace before further processing. However, whitespace will still be 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 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, lazy evaluation may still cause the evaluation order to be not as expected.)
- **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 thereby gain privileged access to resources on your machine.
- **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 determines whether source documents should be parsed with schema-validation enabled.
  - **Lax schema validation ("lax")** - This mode determines whether source documents should be parsed with schema-validation enabled if an XML Schema is provided.
  - **Disable schema validation** - This determines whether 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.

Saxon HE/PE/EE Advanced

The **Saxon HE/PE/EE Advanced** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon HE/PE/EE** > **Advanced** .

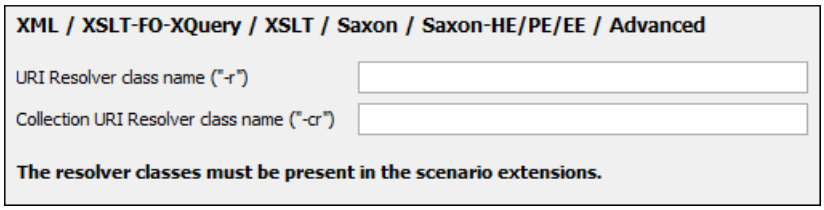


Figure 402: The Saxon HE/PE/EE XSLT Advanced Preferences Panel

There are some advanced XSLT options which can be configured for the Saxon 9.3.0.5 transformer (all three editions: Home Edition, Professional Edition, Enterprise Edition):

- **URI Resolver class name ("-r")** - Allows the user to specify a custom implementation for the URI resolver used by the XSLT Saxon 9.3.0.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from *the dialog for configuring the XSLT extension* for the particular transformation scenario.
- **Collection URI Resolver class name ("-cr")** - Allows the user to specify a custom implementation for the Collection URI resolver used by the XSLT Saxon 9.3.0.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from *the dialog for configuring the XSLT extension* for the particular transformation scenario.

XSLTProc

The **XSLTProc** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** > **XSLTProc** .

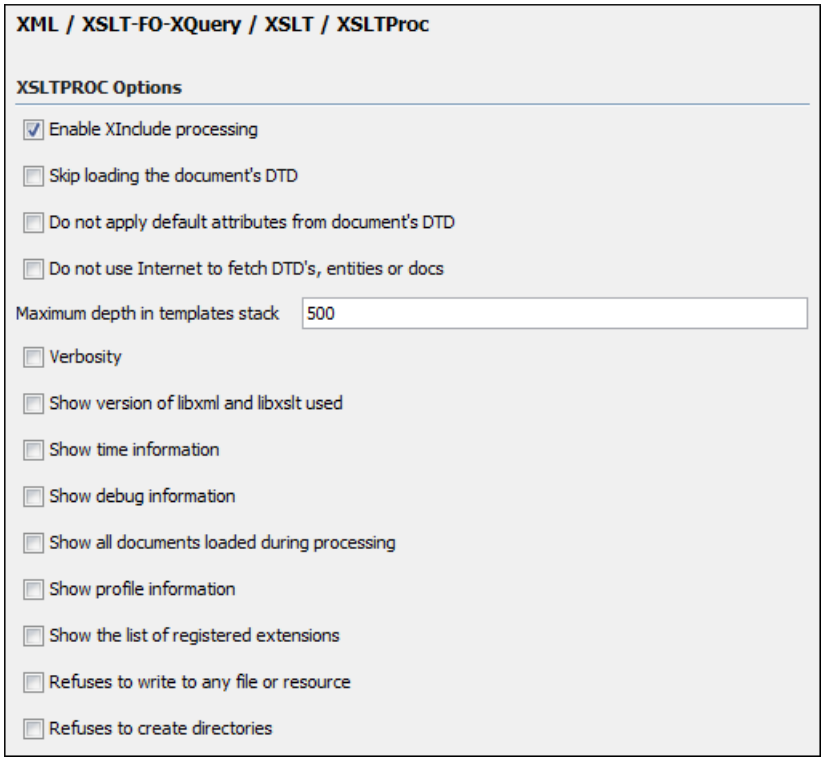


Figure 403: The XSLTProc Preferences Panel

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 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 will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen 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 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

The MSXML preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** > **MSXML** .

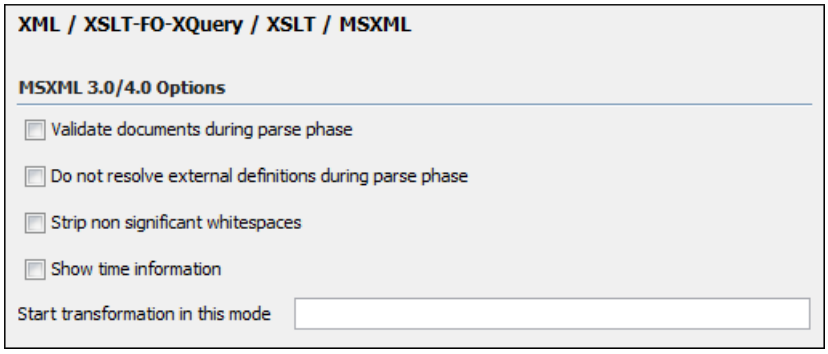


Figure 404: The MSXML Preferences Panel

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

The **MSXML.NET** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XSLT** > **MSXML.NET** .

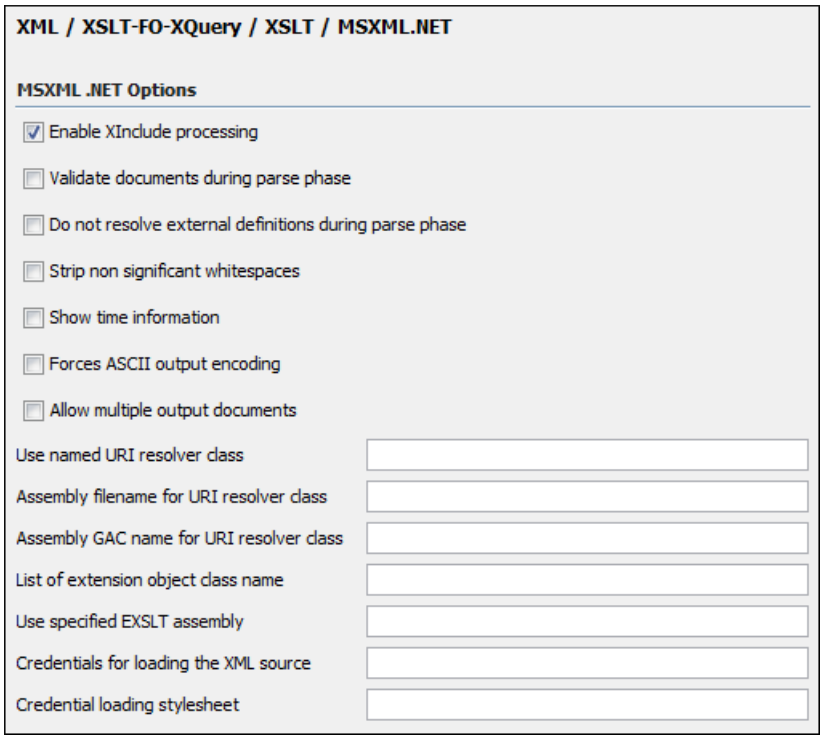


Figure 405: The MSXML.NET Preferences Panel

The options of the MSXML.NET processor are the same as *the ones available in the command line for the MSXML.NET processor*:

- **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 as *when 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

The **XQuery** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XQuery** .

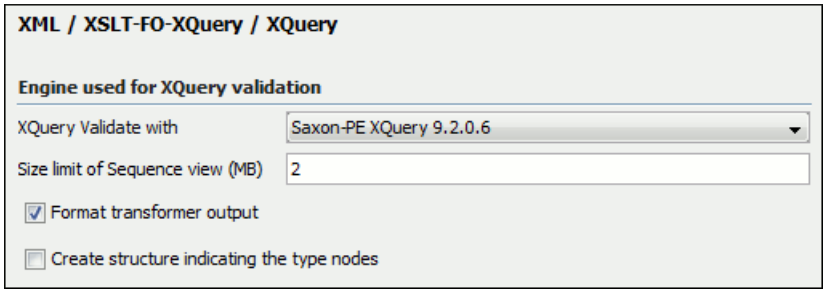


Figure 406: The XQuery Preferences Panel

The generic XQuery preferences are the following:

- **XQuery validate with** - Allows you to select the processor to validate the XQuery. In case you are validating an XQuery file that has an associated validation scenario, Oxygen 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 will be used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor
- **Size limit of Sequence view (MB)** - When the result of an XQuery transformation is *set in the transformation scenario as sequence* the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceed you can extract more data from the database by a click on the **More result available** node from the **Sequence** view.
- **Format transformer output** - When checked the transformer's output is formatted and indented (pretty printed). The option is ignored if in the transformation scenario you choose **Sequence** (lazy extract data from a database).

- **Create structure indicating the type nodes** - If checked, Oxygen takes the results of a query and creates an XML document containing copies of all items in the sequence, suitably wrapped. The option is ignored if in the transformation scenario you choose **Sequence** (lazy extract data from a database).

Saxon HE/PE/EE

The **Saxon HE/PE/EE** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XQuery** > **Saxon HE/PE/EE** .

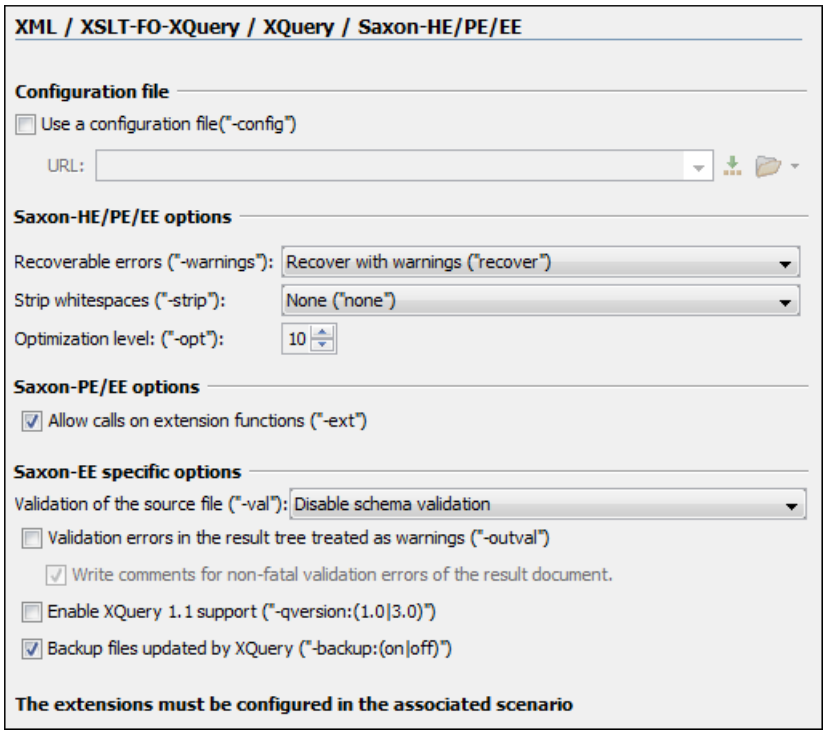


Figure 407: The Saxon XQuery Preferences panel

The XQuery preferences for the Saxon 9.3.0.5 are the following:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that will be used for XQuery transformation and validation.
- **Handling of recoverable stylesheet errors** - Allows the user to choose how dynamic errors will be 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.

The Saxon 9.3.0.5 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 9.3.0.5 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.
- **Enable XQuery 1.1 support** - If it is checked Saxon EE runs the XQuery transformation with the XQuery 1.1 support.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, a backup version is generated for any XML files that is updated with XQuery Update.

Saxon HE/PE/EE Advanced

The **Saxon HE/PE/EE Advanced** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XQuery** > **Saxon HE/PE/EE** > **Advanced** .

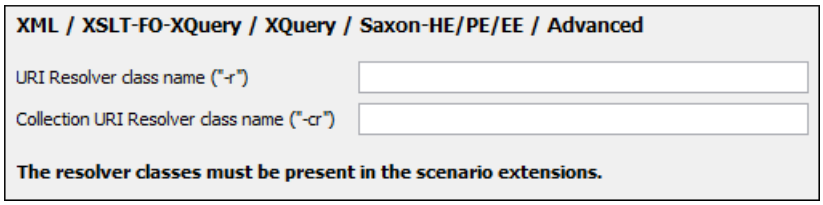


Figure 408: The Saxon HE/PE/EE XQuery Advanced Preferences Panel

The advanced XQuery options which can be configured for the Saxon 9.3.0.5 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 9.3.0.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from *the dialog for configuring the XQuery extension* for the particular transformation scenario.
- **Collection URI Resolver class name** - Allows the user to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.3.0.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from *the dialog for configuring the XQuery extension* for the particular transformation scenario.

Debugger

This section explains the settings available for the Debugger perspective. The settings are available from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **Debugger** .

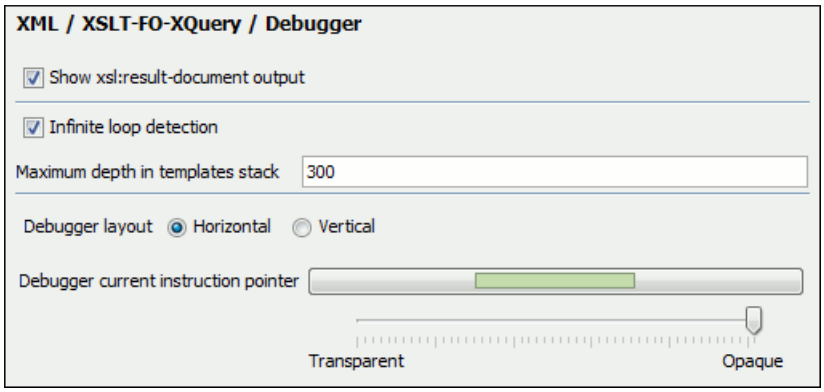


Figure 409: The Debugger Preferences Panel

The debugger preferences are the following:

- **Show xsl:result-document output** - If checked, the debugger presents the output of `xsl:result-document` instructions into the debugger output view.
- **Infinite loop detection** - Set this option to receive notifications when an infinite loop occurs during transformation.
- **Maximum depth in templates stack** - Sets how many `xsl:template` instructions can appear on the current stack. This setting is used by the infinite loop detection.
- **Debugger layout** - A horizontal layout means that the stack of XML editors takes the left half of the editing area and the stack of XSL editors takes the right one. A vertical layout means that the stack of XML editors takes the upper half of the editing area and the stack of XSL editors takes the lower one.
- **Debugger current instruction pointer** - Controls the background color of the current execution node, both in the document (XML) and XSLT/XQuery views.

Profiler

This section explains the settings available for the XSLT Profiler. To access and modify them please go to menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **Profiler** (see *Debugger* on page 690).

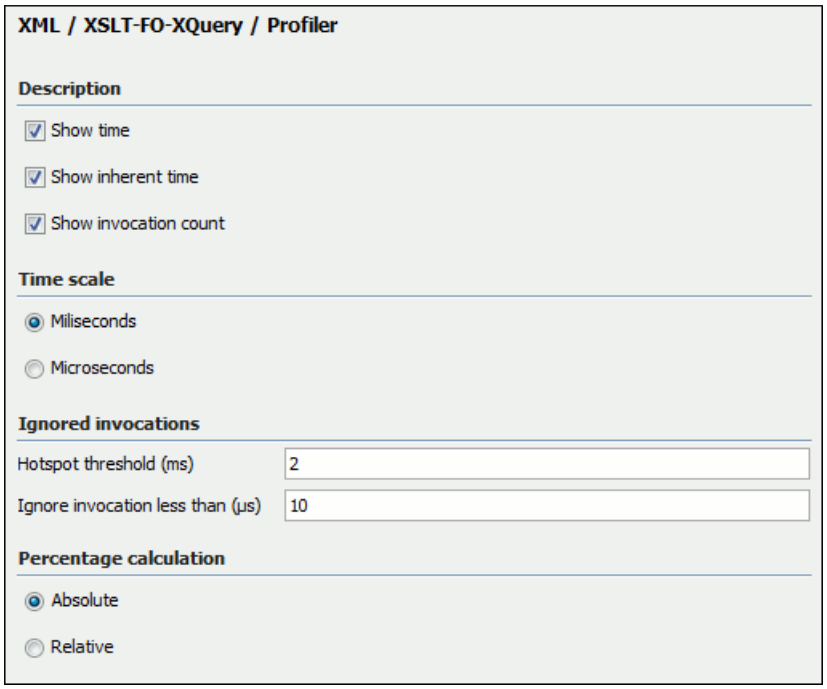


Figure 410: The Profiler Preferences Panel

The following profiles settings are available:

- **Show time** - Shows the total time that was spent in the node.
- **Show inherent time** - Shows the inherent time that was spent in the node. The inherent time is defined as the total time of a node minus the time of its child nodes.
- **Show invocation count** - Shows how many times the node 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 node.

FO Processors

Besides the built-in formatting objects processor (Apache FOP) other external processors can be configured and set in transformation scenarios for processing XSL-FO documents.

Oxygen has implemented 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 external FO processor if the user has the XEP installed. Also, if you have the Antenna House XSL Formatter v4 or v5, Oxygen 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 just as you would for XEP.

The **FO Processors** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **FO Processors** .

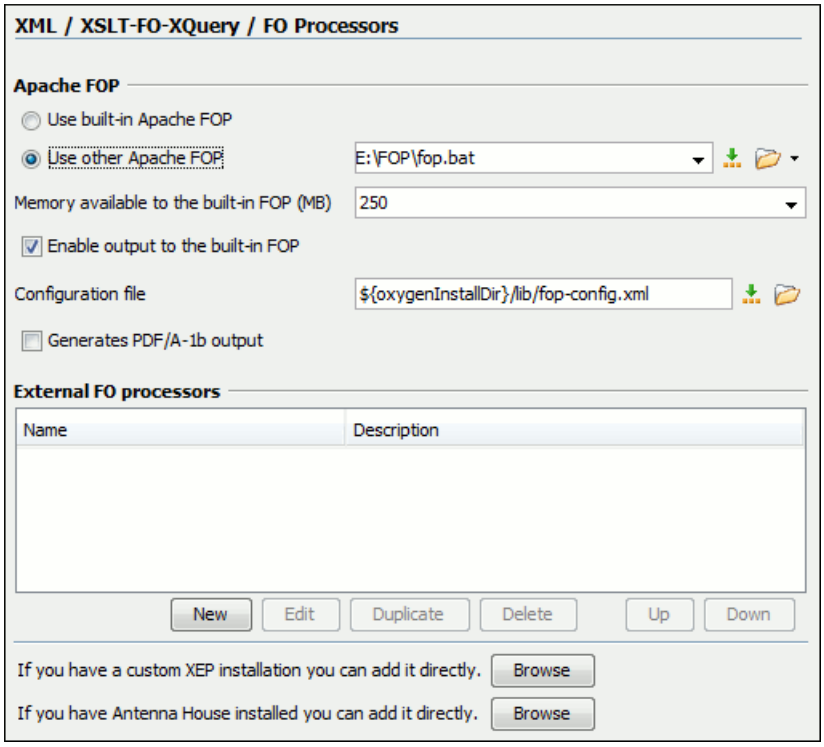


Figure 411: The FO Processors Preferences Panel

The options for FO processors are the following:

- **Use built-in Apache FOP** - Instructs Oxygen to use its built-in Apache FO processor.
- **Use other Apache FOP** - Instructs Oxygen 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 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.*

The users can configure the external FO processors for use in transformation scenarios in the following dialog:

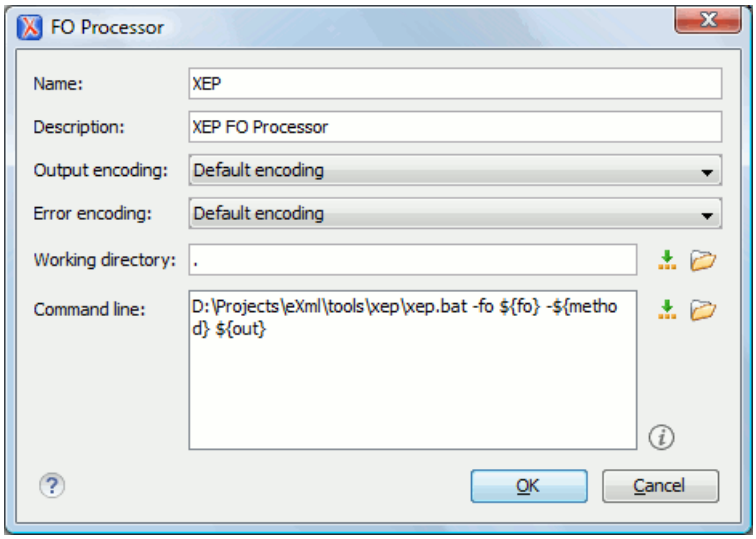


Figure 412: The External FO Processor Configuration Dialog

- **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 window.
- **Error Encoding** - The encoding of the FO processor error stream displayed in a results panel at the bottom of the Oxygen 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 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.
  - **${frameworksDir}** - The path of the `frameworks` subdirectory of the Oxygen install directory.
  - **${oxygenInstallDir}** - The Oxygen 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

The **XPath** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **XPath** .

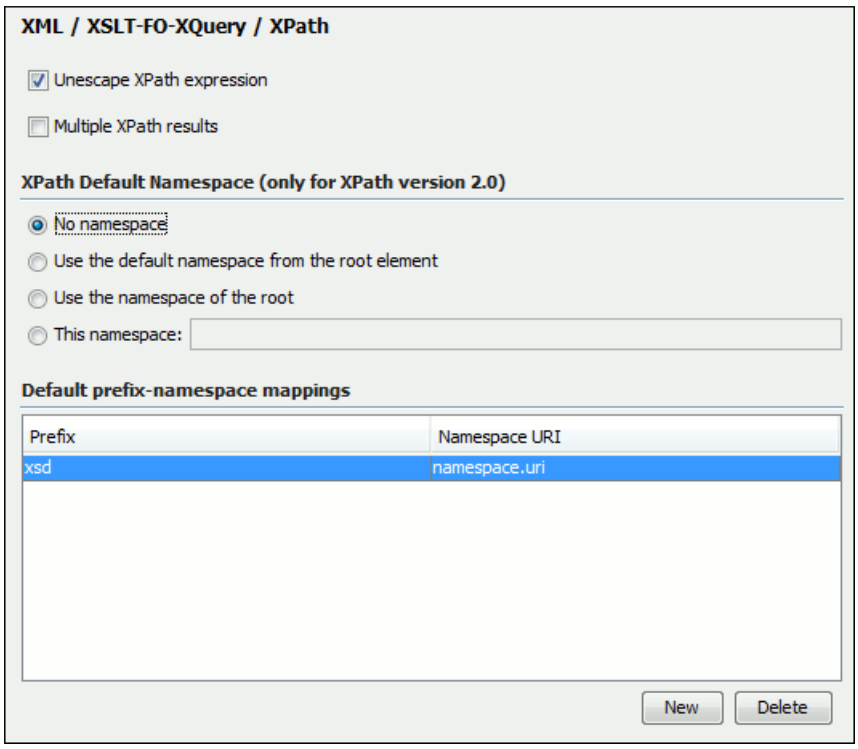


Figure 413: The XPath Preferences Panel

The XPath options are the following:

- **Unescape XPath expression** - When checked, the entities are unescaped in the XPath expressions entered in *the XPath toolbar*. For example the expression

  ```
  //varlistentry[starts-with(@os,'&#x73;')]
  ```

  is equivalent with:

  ```
  //varlistentry[starts-with(@os,'s')]
  ```

- **Multiple XPath results** - If checked, results of different XPath expressions executed on the same file are displayed in separate result set tabs.
- **No namespace** - If checked, Oxygen will consider unprefixed element names in XPath 2.0 expressions evaluated in *the XPath console* as belonging to no namespace.
- **Use the default namespace from the root element** - If checked, Oxygen will consider unprefixed element names in XPath expressions evaluated in *the XPath console* as belonging to the default namespace declared on the root element of the queried XML document.
- **Use the namespace of the root** - If checked, Oxygen will consider unprefixed element names in XPath expressions evaluated in *the XPath console* as belonging to the same namespace as the root element of the document.
- **This namespace** - The user has the possibility to enter here the namespace of the unprefixed elements used in *the XPath console*.
- **Default prefix-namespace mappings** - Associates prefixes to namespaces. These mappings are useful when applying an XPath in the XPath console and you don't want to define these mappings in each document separately.

Custom Engines

You can configure and run XSLT and XQuery transformations with processors other than *the ones which come with the Oxygen distribution*. Such an external engine can be used in the Editor perspective and is available in the list of engines in *the dialog for editing a transformation scenario*. However it cannot be used in *the Debugger perspective*.

The **Custom Engines** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **Custom Engines** .

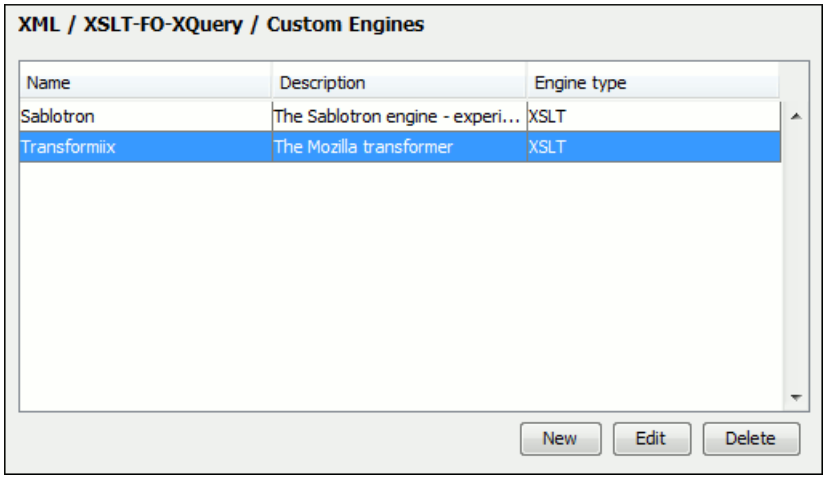


Figure 414: Custom Transformation Engines

The following parameters can be configured for a custom engine:

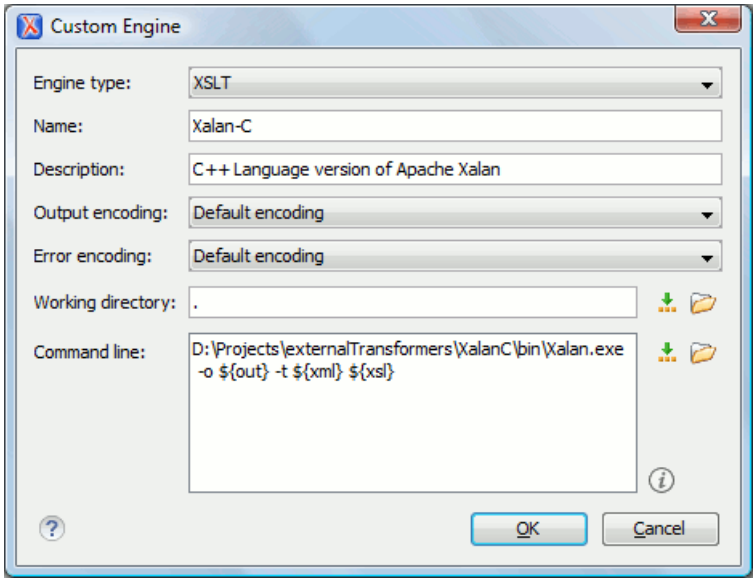


Figure 415: Parameters of a Custom Engine

- **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 install directory.
- **Command line** - The command line that must be executed by Oxygen 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

The **Import** preferences panel is opened from menu **Options** > **Preferences** > **XML** > **Import** . Here you can configure how empty values and null values are handled when they are encountered in an imported database tables or Excel sheets. Also you can configure here the format of date / time values that should be recognized in the imported database tables or Excel sheets.

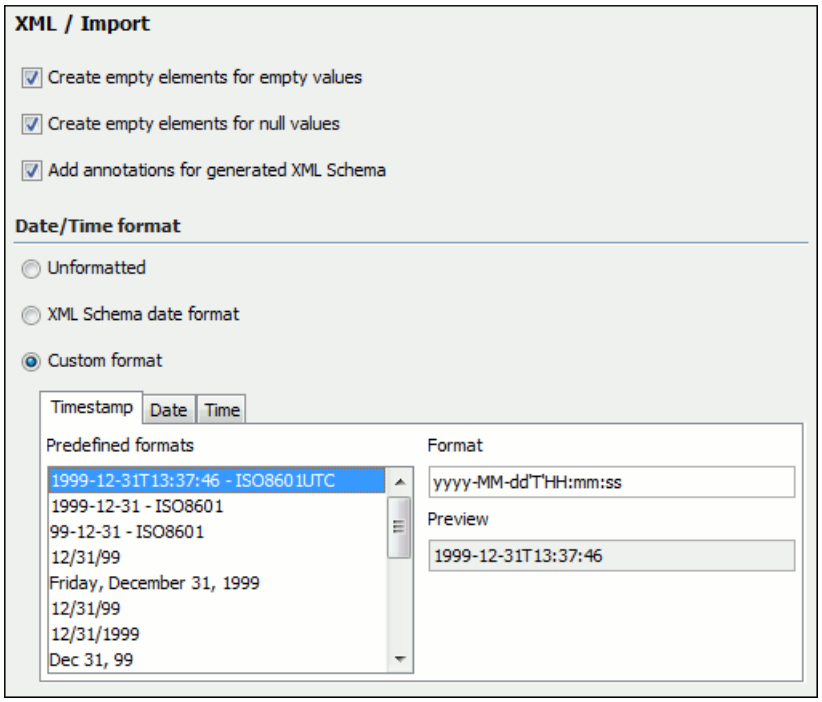


Figure 416: The XML Import Preferences Panel

The import preferences are the following:

- **Create empty elements for empty values** - If checked, an empty value from a database column or from a text file will be imported as an empty element.
- **Create empty elements for null values** - If this checked, a null value from a database column will be imported as an empty element.
- **Add annotations for generated XML Schema** - If checked, the generated XML Schema will contain an annotation for each of the imported table's columns. The documentation inside the annotation tag will contain 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 will be used for import. When importing data from Excel a string representation of date or time values will be used. The type used in the generated XML Schema will be `xs:string`.

- **XML Schema date format** - If checked, the XML Schema specific format ISO8601 will be 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 will be `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 from 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

Table 11: 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
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

The **Data Sources** preferences panel is opened from menu **Options** > **Preferences** > **Data Sources** .

Configuration of Data Sources

Here 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.

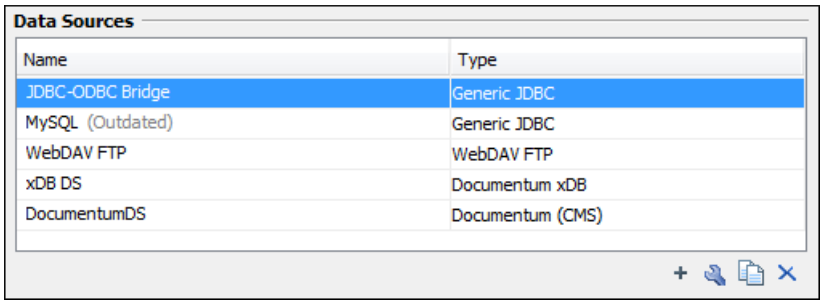


Figure 417: The Data Sources Preferences Panel

- **New** - Opens the **Data Sources Drivers** dialog that allows you to configure a new database driver.

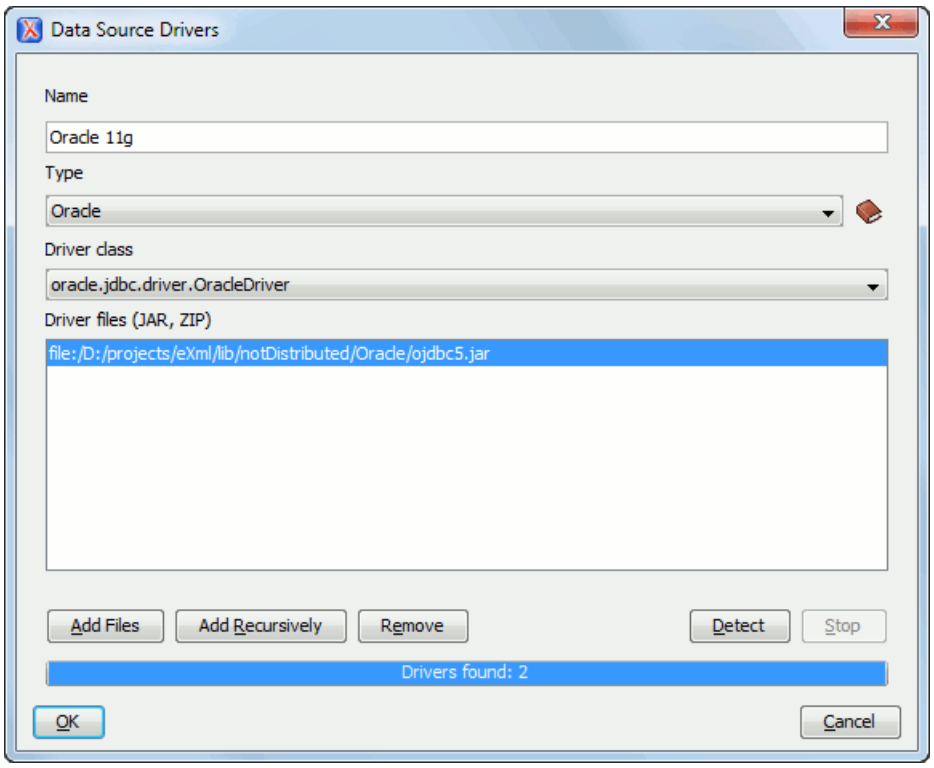


Figure 418: The Data Sources Drivers Dialog

The fields of the dialog are the following:

- **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.

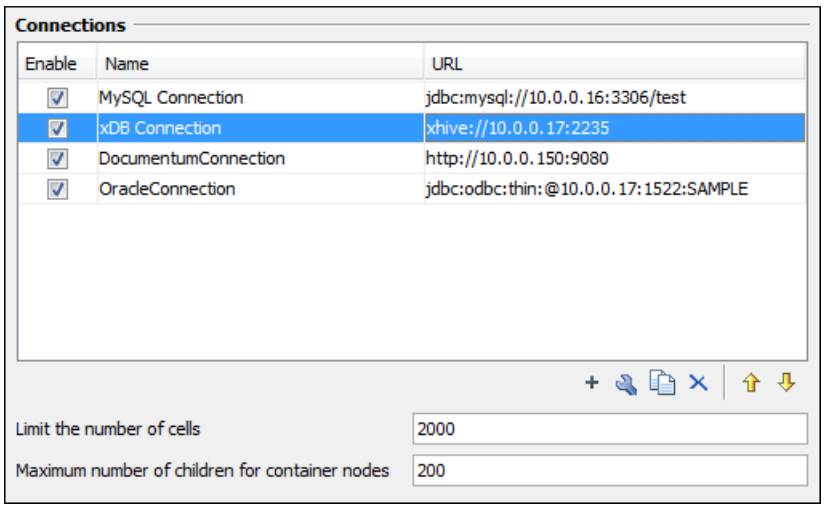


Figure 419: The Connections Preferences Panel

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 and Tamino databases a container can hold millions of resources. If the node corresponding to such a container in the **Data Source Explorer** view would display all the contained resources at the same time the performance of the view would be very slow. To prevent such a situation only a limited number of the contained resources is displayed as child nodes of the container node. Navigation to other contained resources from the same container is enabled by the *Up* and *Down* buttons in the **Data Source Explorer** view. This limited number is set in the option **Maximum number of children for container nodes**. The default value is 200 nodes.

The actions of the buttons from the **Connections** panel are the following:

- **New** - Opens the **Connection** dialog which has the following fields:

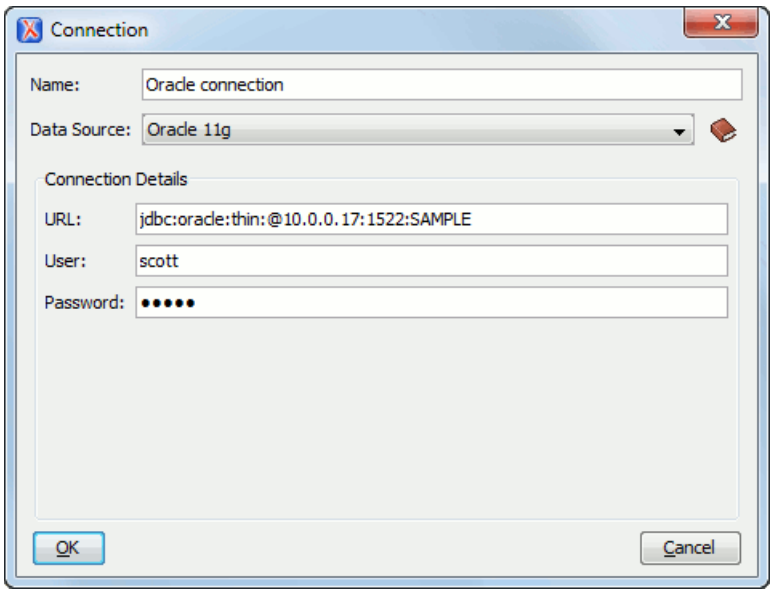


Figure 420: The Connection Dialog

  - **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).

- **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.

- **Berkeley DB XML database** - Copy the jar files from the Berkeley database install directory to the Oxygen 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 for *configuring a DB2 data source*.
- **eXist database** - Copy the jar files from the eXist database install directory to the Oxygen install directory as described in *the procedure* for configuring an eXist data source.
- **MarkLogic database** - Download the Java and .NET XCC distributions (XCC Connectivity Packages) from *MarkLogic*. You find the details about configuring a MarkLogic data source in *the procedure for creating a MarkLogic data source*.
- **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 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*.
- **RainingData TigerLogic XDMS database** - Copy the jar files from the TigerLogic JDK `lib` directory from the server side to the Oxygen install directory as described in *the procedure* for configuring a TigerLogic data source.
- **SoftwareAG Tamino database** - Copy the jar files from the `SDK\TaminoAPI4J\lib` subdirectory of the Tamino database install directory to the Oxygen install directory as described in *the procedure* for configuring a Tamino 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 install directory as described in *the procedure* for configuring a Documentum xDb (X-Hive/DB) 10 data source.

- **MySQL database** - Download the MySQL JDBC driver from *the MySQL website* and use it for *configuring a MySQL data source*.

Table Filters

The **Table Filters** preferences panel is opened from menu **Options** > **Preferences** > **Data Sources** > **Table Filters** .

Here you can choose which of the database table types will be displayed in the **Data Source Explorer** view.

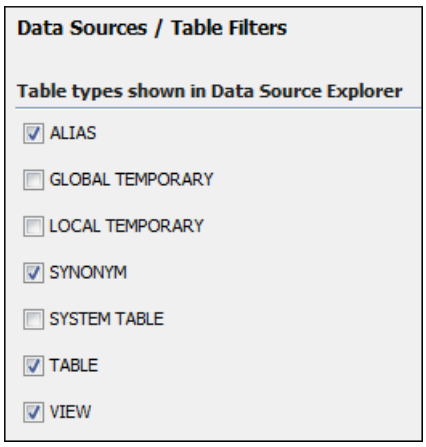


Figure 421: Table Filters Preferences Panel

SVN

The **SVN** preferences panel is opened from menu **Options** > **Preferences** > **SVN** and it is the place where the user preferences for the embedded SVN client tool are configured. Some other preferences for the embedded SVN client tool can be set in the global files called `config` and `servers`, that is the files with parameters that act as defaults applied to all the SVN client tools that are used by the same user on his login account on the computer. These files can be opened for editing with the two edit actions available in the SVN client tool on the **Global Runtime Configuration** submenu of the **Options** menu.

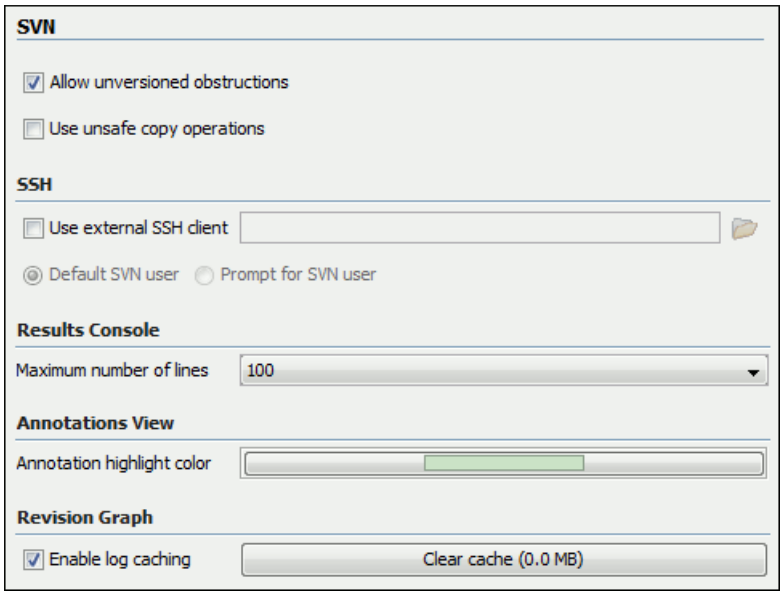


Figure 422: The SVN Preferences Panel

The SVN preferences are the following:

- **Enable symbolic link support** (*available only on Mac OS X and Linux*) - Subversion has the ability to put a symbolic link under version control, via the usual SVN `add` command. The Subversion repository has no internal concept of a symbolic link, it stores a versioned symbolic link as an ordinary file with a `svn:special` property attached. The SVN client (on Unix) sees the property and translates the file into a symbolic link in the working copy.

  👉 **Note:** Windows file systems have no symbolic links, so a Windows client won't do any such translation: the object appears as a normal file.

  If the symbolic link support is disabled then the versioned symbolic links, on Linux and OS X, are supported in the same way as on Windows, that is a text file instead of symbolic link is created.

  👉 **Important:** It is recommended to disable symbolic links support if you do not have versioned symbolic links in your repository, because the SVN operations will work faster. However, you should not disable this option when you do have versioned symbolic links in repository. In that case a workaround would be to refer to working copy by its real path, not a path that includes a symbolic link.

- **Allow unversioned obstructions** - This option controls how should be handled working copy resources being ignored / unversioned when performing an update operation and from the repository are incoming files with the same name, in the same location, that intersect with those being ignored / unversioned. If the option is enabled, then the incoming items will become BASE revisions of the ones already present in the working copy, and those present will be made versioned resources and will be marked as modified. Exactly as if the user first made the update operation and after that he / she modified the files. If the option is disabled, the update operation will fail when encountering files in this situation, possibly leaving other files not updated. By default, this option is enabled.
- **Use unsafe copy operations** - Sometimes when the working copy is accessed through Samba and SVN client cannot make a safe copy of the committed file due to a delay in getting write permission the result is that the committed file will be saved with zero length (the content is removed) and an error will be reported. In this case this option should be selected so that SVN client does not try to make the safe copy.
- **SSH** - Here you can specify the command line for an external SSH client which will be used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments (if any). Depending on the SSH client used and your SSH server configuration you may need to specify in the command line the user name and / or private key / passphrase. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example on Windows the following command line uses the `plink.exe` tool as external SSH client for connecting to the SVN repository with SVN+SSH:

  ```
  C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
  host_name_or_IP_address_of_SVN_server
  ```

- **Results Console** - Here you can specify the maximum number of lines displayed in the **Console** view.
- **Annotations View** - Here you can set the color used for highlighting in the editor panel all the changes contributed to a resource by the revision selected in *the **Annotations** view*.
- **Revision Graph** - Here you can enable caching for the action of computing a revision graph. When a new revision graph is requested one of the caches from the previous actions may be used which will avoid running the whole query again on the SVN server. If a cache is used it will finish the action much faster.

Working Copy

The **Working Copy** panel is open from menu **Options** > **Preferences** > **SVN** > **Working Copy** and it contains options that are specific to SVN working copies.

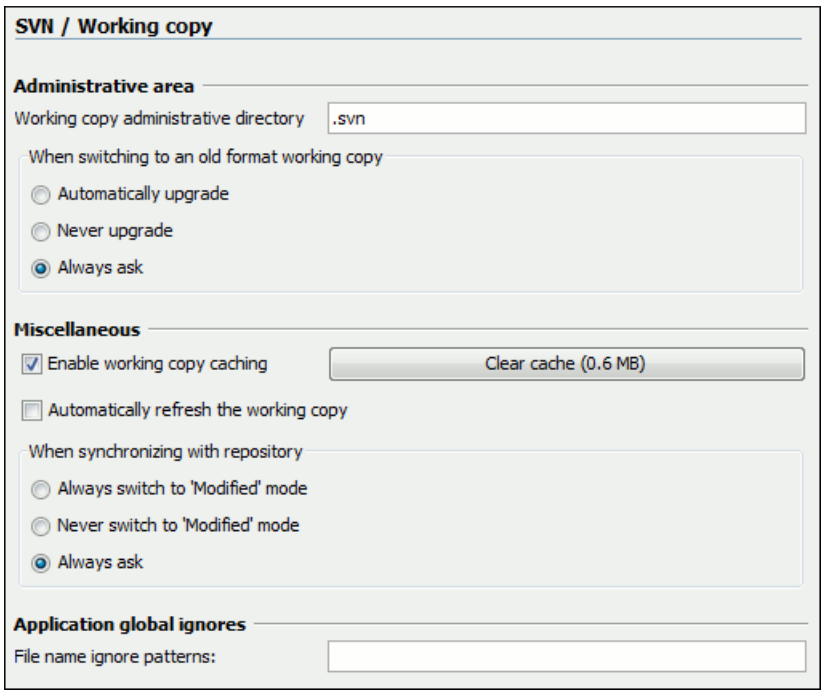


Figure 423: The Working Copy Panel

These options are the following:

- **Working copy administrative directory** - Allows you to customize the directory name where the svn entries are kept for each directory in the working copy.
- **When switching to an old format working copy** - You can instruct Syncro SVN Client to do one of the following:
  - **Automatically upgrade** - Older format working copies are upgraded to the newest known format.
  - **Never upgrade** - Older format working copies are left untouched. No attempt to upgrade the format is made.
  - **Always ask** - You are notified when such a working copy is used and you are allowed to choose what action to be taken - to upgrade or not the format of the current working copy.
- **Enable working copy caching** - If checked, the content of the working copies is cached for refresh operations.
- **Automatically refresh the working copy** - If checked, the working copy is refreshed from cache. Only the new changes (modifications with a date/time that follows the last refresh operation) are refreshed from disk. Disabled by default.
- **When synchronizing with repository** - The action that will be executed automatically after the **Synchronize** action. The possible actions are:
  - **Always switch to 'Modified' mode** - The **Synchronize** action is followed automatically by a switch to **Modified** mode of **Working Copy** view, if **All Files** mode is currently selected.
  - **Never switch to 'Modified' mode** - Keeps the currently selected view mode unchanged.
  - **Always ask** - The user is always asked if he wants to switch to **Modified** mode.
- **Application global ignores** - Allows setting file patterns that may include the * and ? wildcards for unversioned files and folders that must be ignored when displaying the working copy resources in *the **Working Copy** view*.

Diff

The **Diff** preferences panel is opened from menu **Options** > **Preferences** > **SVN** > **Diff** and it allows you to set the compare options for SVN client.

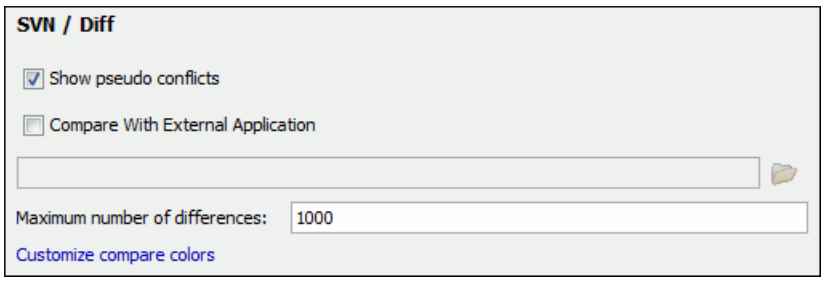


Figure 424: The SVN Diff Preferences Panel

The SVN diff preferences are the following:

- **Show pseudo conflicts** - It allows you to specify if you want to see pseudo-conflicts in *the **Compare** view*. A pseudo conflict occurs when two developers make the same change, for example when both add or remove the same line of code.
- **Compare With External Application** - You can specify an external application to be launched for compare operations in the following cases:
  - when two history revisions are compared
  - when the working copy file is compared with a history revision
  - when *a conflict is edited*

  The parameters ${firstFile} and ${secondFile} specify the positions of the two compared files in the command line for the external diff application. The parameter ${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison: the working copy version of a file is compared with the repository version, with the BASE revision (the latest revision read from the repository by an Update or Synchronize operation) being the common ancestor of these two compared versions.
- **Maximum number of differences** - You can change the maximum number of differences allowed in the view.

Messages

The **Messages** preferences panel is opened from menu **Options** > **Preferences** > **SVN** > **Messages** and allows disabling the following warning messages which may appear in the application:

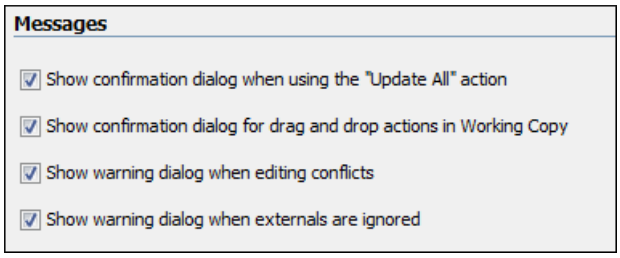


Figure 425: The Messages Preferences Panel

- **Show confirmation dialog when using the "Update All" action** - Allows you to avoid performing accidental update operations by requesting you to confirm them before execution.
- **Show confirmation dialog for drag and drop actions in Working Copy** - This option avoids doing a drag and drop when you just want to select multiple files in the Working Copy view.
- **Show warning dialog when editing conflicts** - When the **Edit Conflicts** action is executed, a warning dialog notifies you that the action overwrites the conflicted version of the file created by an update operation. The conflicted file is overwritten with the version of the same file which existed in the working copy before the update operation and then *proceeds with the visual editing of the conflicting file*.
- **Show warning dialog when "svn:externals" definitions are ignored** - A warning dialog is displayed when "svn:externals" definitions are ignored before performing any operation that updates resources of the working copy (like *Update* and *Override and Update*).

Files Comparison

Oxygen XML Editor offers six different diff algorithms to choose from for file comparison:

- two XML diff algorithms: **XML Fast** and **XML Accurate**,
- a **Syntax Aware** algorithm that gives very good results on all file types known by Oxygen XML Editor ,
- three all-purpose algorithms: line based, word based and character based.

Any of these six algorithms can be used to perform differences on request, but Oxygen XML Editor offers also an automatic mode that selects the most appropriate one, based on the file content and size.

The **Files Comparison** preferences panel is opened from menu **Options** > **Preferences** > **Diff** > **Files Comparison** and offers the following configurable preferences:

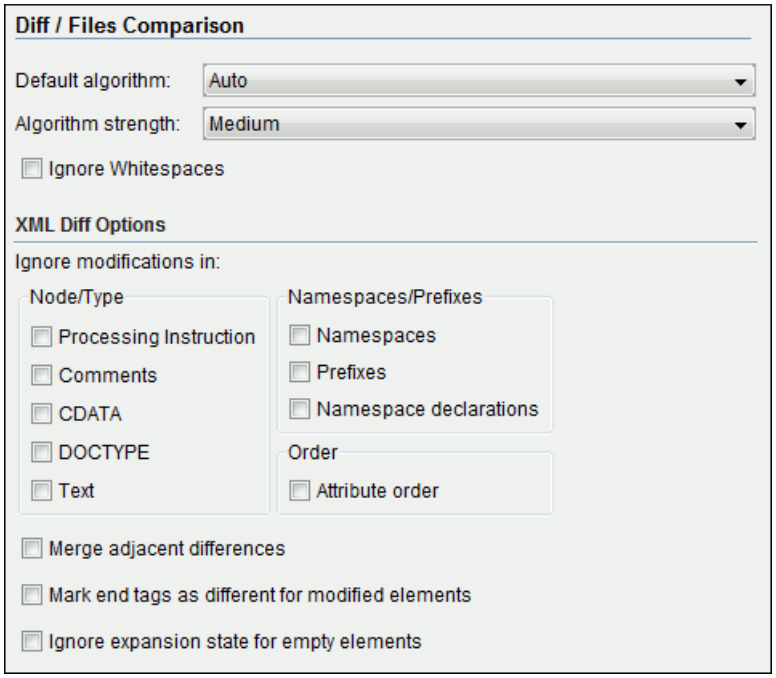


Figure 426: The Files Comparison Preferences Panel

- **Default algorithm** - The default algorithm used for comparing files. The following options are available:
  - **Auto** - Automatic selection of the diff algorithm, based on the file content and size.
  - **Characters** - Computes the differences at character level.
  - **Words** - Computes the differences at word level.
  - **Lines** - Computes the differences at line level.
  - **Syntax aware** - For the file types known by Oxygen XML Editor , this algorithm computes the differences taking into consideration the syntax (the specific types of tokens) of the documents.
  - **XML Fast** - A diff algorithm well-suited for large XML documents (tens of MB). Sacrifices accuracy in favor of speed.
  - **XML Accurate** - XML-tuned diff algorithm. It favors accuracy over speed.
- **Algorithm strength** - Controls the amount of resources allocated to the application to perform the comparison. The algorithm stops searching more differences when reaches the maximum allowed resources. A dialog is shown when this limit is reached and partial results are displayed. Four settings are available: **Low**, **Medium** (default), **High** and **Very High**.
- **Ignore Whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.

- **XML Diff Options** - This set of options allows you to specify the types of XML nodes for which the differences will be ignored (will not be reported) by the **XML Fast** and **XML Accurate** algorithms.
- **Merge adjacent differences** - If checked, it considers two adjacent differences as one when the differences are painted in the side-by-side editors. If unchecked, every difference is represented separately.
- **Mark end tags as different for modified elements** - If checked, end tags of modified elements are presented as differences too, otherwise only the start tags are presented as differences.
- **Ignore expansion state for empty elements** - If checked, empty elements in both expansion states are considered matched, that is `<element/>` and `<element></element>` are considered equal.

Appearance

The **Files Comparison / Appearance** preferences panel is opened from menu **Options** > **Preferences** > **Diff** > **Files Comparison** > **Appearance** and offers the following options:

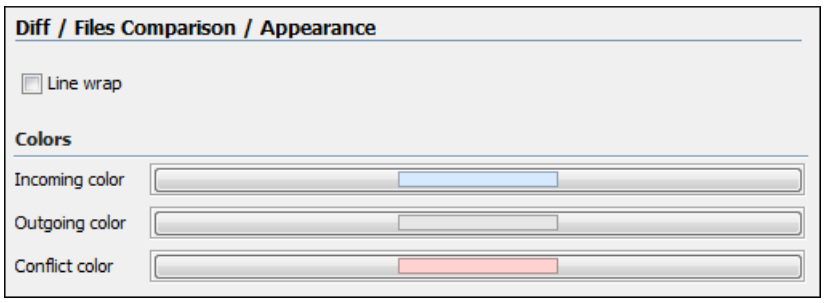


Figure 427: Files Comparison Appearance Preferences Panel

- **Line wrap** - If checked, the lines presented in the two diff panels are wrapped at the right margin of each panel so that no horizontal scrollbar is necessary.
- **Incoming color** - The color used for incoming changes on the vertical bar that shows the differences between the files compared.
- **Outgoing color** - The color used for outgoing changes on the vertical bar that shows the differences between the files compared.
- **Conflict color** - The color used for conflicts on the vertical bar that shows the differences between the files compared.

Directories Comparison

The **Directories Comparison** preferences panel is opened from menu **Options** > **Preferences** > **Diff** > **Directories Comparison** and offers the following configurable preferences:

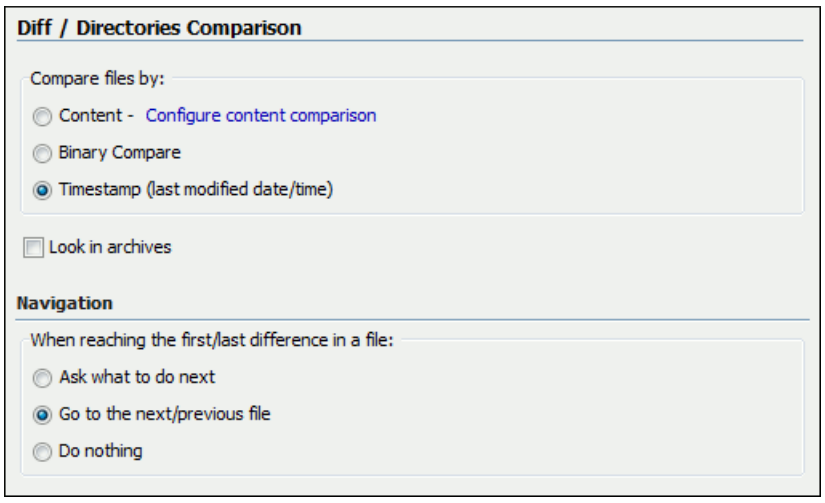


Figure 428: The Diff Preferences Panel

For the directories comparison, you can specify:

- **Compare files by** - Controls the method used for comparing two files:
  - **Content** - The file content is compared using the current *diff algorithm*. This option is applied for a pair of files only if that file type is associated with a built-in editor type (either associated by default or associated by the user when the user is prompted to do that on opening a file of that type for the first time).
  - **Binary Compare** - The files are compared at byte level.
  - **Timestamp (last modified date / time)** - The files are compared only by their last modified timestamp.
- **Look in archives** - If checked, *archives known by Oxygen XML Editor* are considered directories and their content is compared just like regular files.
- **Navigation** - This options control the behaviour of the differences traversal actions (**Go to previous modification**, **Go to next modification**) when the first or last difference in a file is reached:
  - **Ask what to do next** - A dialog is displayed asking you to confirm that you want the application to display modifications from the previous or next file.
  - **Go to the next/previous file** - The application opens the next or previous file without waiting for your confirmation.
  - **Do nothing** - No further action is taken.

Appearance

The **Directories Comparison / Appearance** preferences panel is opened from menu **Options** > **Preferences** > **Diff** > **Directories Comparison** > **Appearance** .

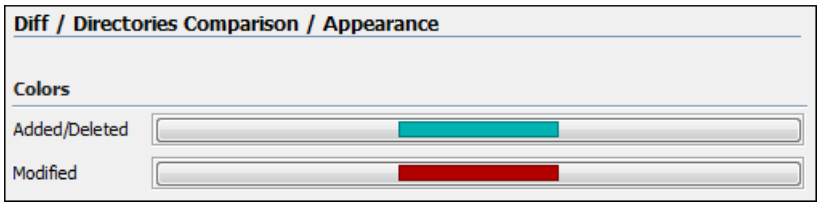


Figure 429: The Diff Appearance Preferences Panel

- **Added/Deleted** - Color used for marking added or deleted files and folders.
- **Modified** - Color used for marking modified files.

Archive

The **Archive** preferences panel is opened from menu **Options** > **Preferences** > **Archive** .

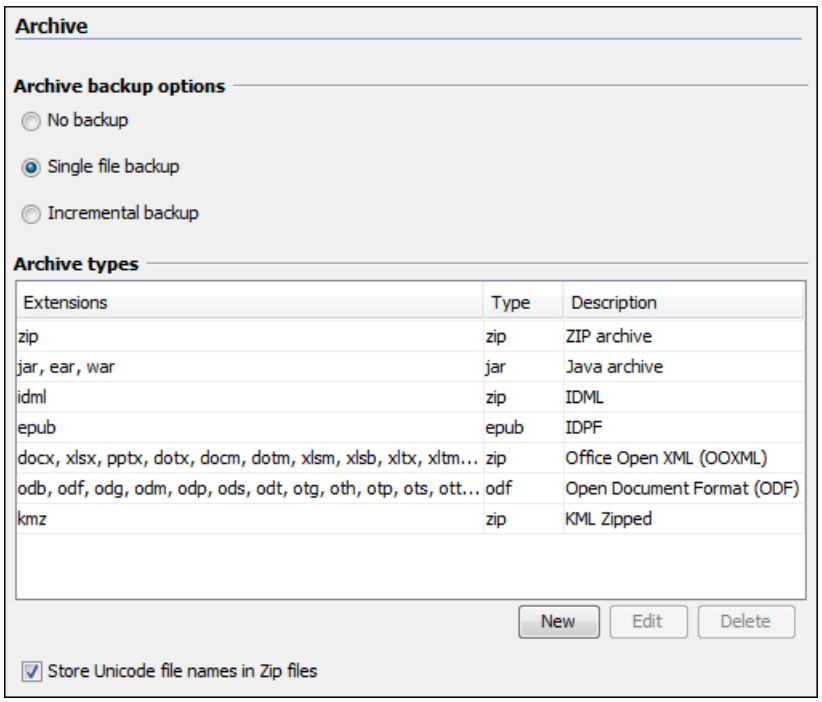


Figure 430: The Archive Preferences Panel

The following options are available in the **Archive** preferences panel:

- One of the following options is the default for backup actions in the **Archive Backup** dialog:
  - **No backup** - No backups are made.
  - **Single file backup** - When you modify an archive, its content is backed up under the name `originalArchiveFileName.bak`. You can find the backup file in the same folder as the original archive.

    Note: The backup is done only once per application session for each archive open in the **Archive Browser** view.

  - **Incremental backup** - When you modify an archive, its content is backed up under the name `originalArchiveFileName.bakNumber`. *Number* is an incremental integer, indicating how many backups were made so far. You can find the backup file in the same folder as the original archive.

    Note: The backup is done only once per application session for each archive open in the **Archive Browser** view.

- **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. You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.

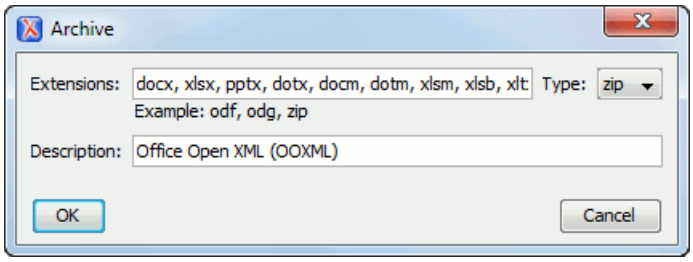


Figure 431: Edit Archive Extension Mappings

Important: You have to restart Oxygen 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.

Plugins

Oxygen provides the ability to add plugins that extend the functionality of the application. The plugins are shipped as separate packages. Check for new plugins *on Oxygen site*.

One plugin consists of a separate sub-folder in the `Plugins` folder of the Oxygen installation folder. This sub-folder must contain a valid `plugin.xml` file in accordance with the `plugin.dtd` file from the `Plugins` folder.

Oxygen automatically detects and loads plugins correctly installed in the `Plugins` folder and displays them in this preferences panel.

The **Plugins** preferences panel is opened from menu **Options** > **Preferences** > **Plugins** .

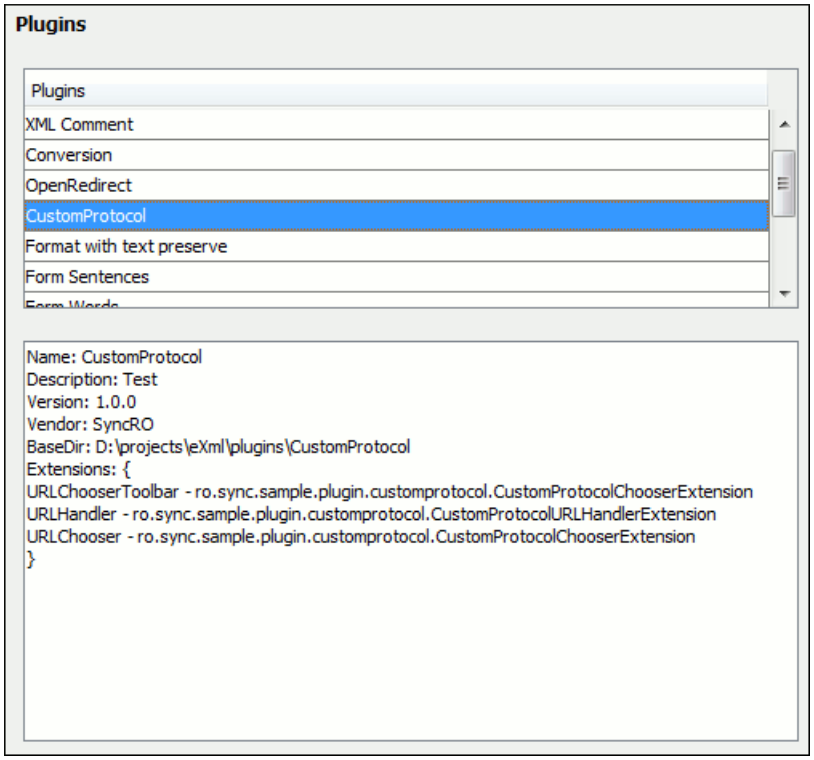


Figure 432: The Plugins Preferences Panel

A list of the plugin properties can be obtained with a click on the plugin name.

External Tools

The **External Tools** preferences panel is opened from menu **Preferences** > **External Tools** .

A command-line tool can be started from the **External Tools** toolbar in the Oxygen user interface as if from the command line of the operating system shell. Such a tool must be configured first in this preferences panel. The parameters of an external tool are set in the following dialog:

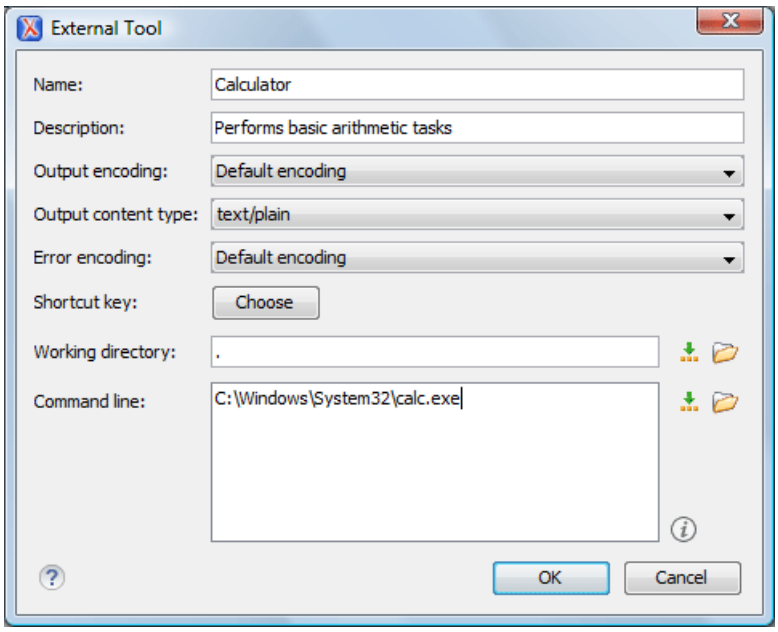


Figure 433: The External Tools Configuration Dialog

- **Name** - The name of the menu entry corresponding to this tool that will be displayed in the **Tools** > **External Tools** menu and in the **External Tools** toolbar.
- **Description** - The description of the tool displayed as tooltip where the tool name is used.
- **Output Encoding** - The encoding of the output stream of the external tool that will be used by Oxygen to read the output of the tool.
- **Output content type** - A list of predefined content type formats that instruct Oxygen how to display the generated output. For example, setting the **Output content type** to `text/xml` enables the syntax coloring of XML output.
- **Error Encoding** - The encoding of the error stream of the external tool that will be used by Oxygen to read this error stream.
- **Shortcut key** - The keyboard shortcut that launches the external tool.
- **Working directory** - The directory the external tool will use to store intermediate and final results. Here you can use one of the following editor variables:
  - **${homeDir}** - The path to user home directory.
  - **${cfd}** - The path of current file directory.
  - **${pd}** - The project directory.
  - **${oxygenInstallDir}** - The Oxygen installation directory.
- **Command line** - The command line that will start the external tool. Here you can use one of the following editor variables:
  - **${dbgXML}** - The path to the current Debugger source selection (for tools started from the XSLT/XQuery Debugger).
  - **${dbgXSL}** - The path to the current Debugger stylesheet selection (for tools started from the XSLT/XQuery Debugger).
  - **${homeDir}** - The path to the user home directory.
  - **${cfn}** - The current file name without extension.
  - **${cfne}** - The current file name with extension.
  - **${cf}** - The path of the currently edited file.
  - **${cfd}** - The path of the current file directory.
  - **${tsf}** - The transformation result file.
  - **${pd}** - The project directory.
  - **${oxygenInstallDir}** - The installation directory of the application.

- **${frameworksDir}** - The directory where the Oxygen frameworks are located.
- **${ps}** - The path separator which can be used on different operating systems between libraries specified in the class path.
- **${timeStamp}** - The current time stamp in Unix format which can be used for example to save transformation results in different output files on each transform.

Menu Shortcut Keys

The **Menu Shortcut Keys** preferences panel is opened from menu **Options** > **Preferences** > **Menu Shortcut Keys** . It allows configuring in one place the keyboard shortcuts available for the menu items on the menus of Oxygen . The current shortcuts assigned to menu items are displayed in the following table.

You can find an operation in the table using the filter field that can search by the operation's description, category or shortcut key:

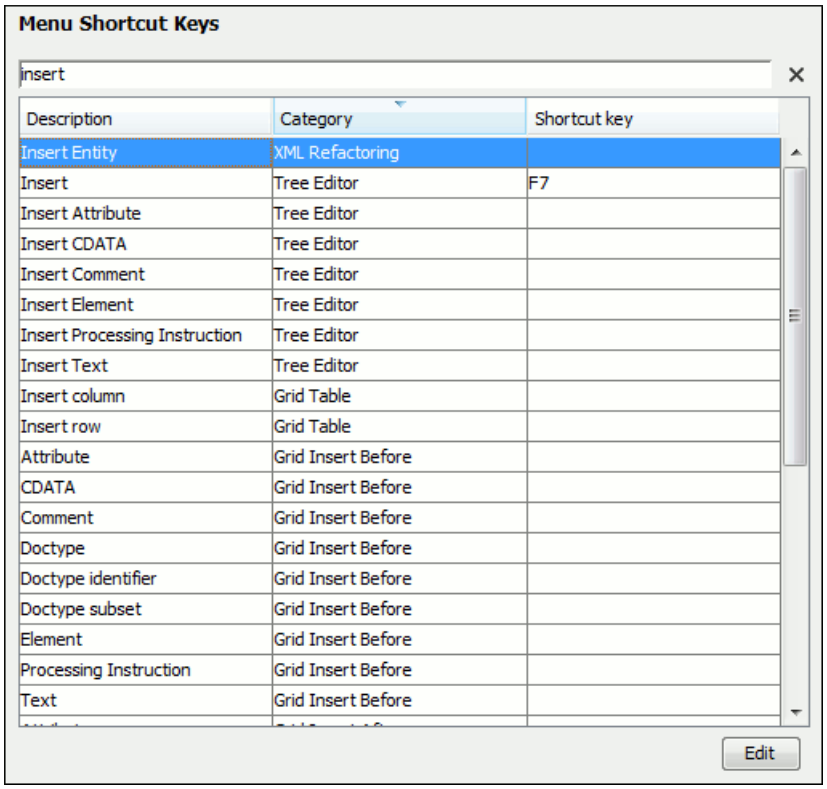


Figure 434: The Menu Shortcut Keys Preferences Panel

- **Description** - A short description of the menu item operation.
- **Category** - The shortcuts are classified in categories for easier management. For example the **Cut** operation for the source view is distinguished from the tree view one by assigning it to a separate category.
- **Shortcut key** - The keyboard shortcut that launches the operation. Double-clicking on a table row or pressing the **Edit** button allows the user to register a new shortcut for the operation displayed on that row.
- **'Home' and 'End' keys are applied at line level** - Option available only on Mac OS X that controls the way the HOME and END keys are interpreted. If checked, the default behaviour of the Mac OS X HOME and END keys will be overridden and the caret will move only on the current line. The default on the Mac is to move the caret to the beginning or end of the document.

File Types

Oxygen XML Editor offers editing support for a wide variety of file types, but users are free to add new file extensions and associate them with the editor type which fits better. The associations set here between a file extension and the type of editor which opens the file for editing have precedence over the default associations available in *the **File > New** dialog*.

The **File Types** preferences panel is opened from menu **Options** > **Preferences** > **File Types** .

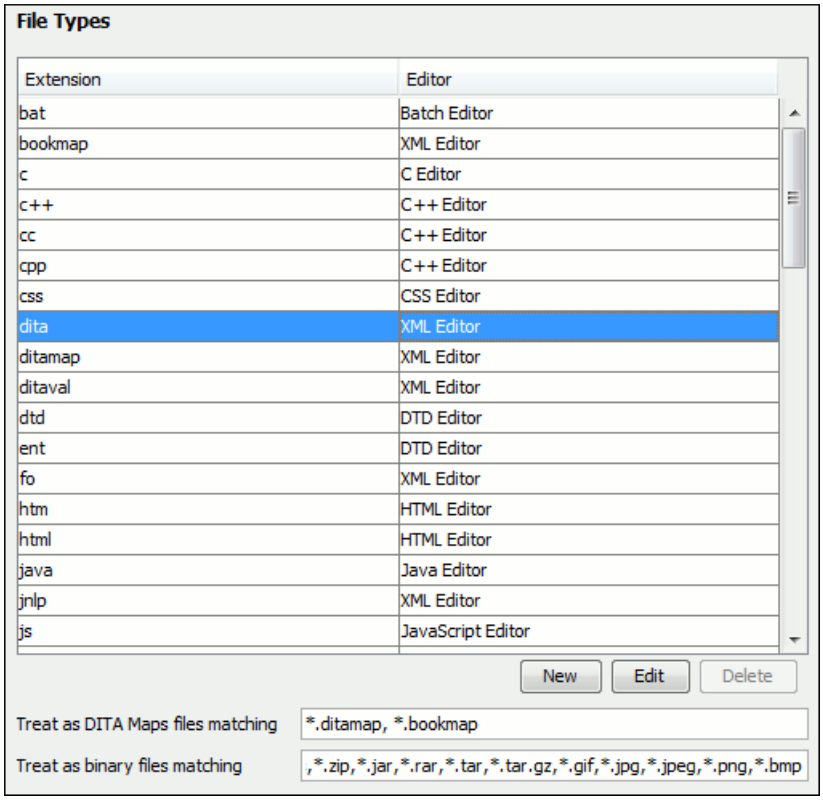


Figure 435: The File Types Preferences Panel

The table columns contain the following data:

- **Extension** - The extensions of the files that will be associated with an editor type.
- **Editor** - The type of editor which the extensions will be associated with. Some editors provide easy access to frequent operations via toolbars (e.g. XML editor, XSL editor, DTD editor) while others provide just a syntax highlight scheme (e.g. Java editor, SQL editor, Shell editor, etc.).

  If the editor set here is not one of the XML editors (XML editor, XSL editor, XSD editor, RNG editor, WSDL editor) then the encoding set in *the preference **Encoding for non XML files*** is used for opening and saving a file of this type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the encoding of the file.

You can also decide the files which will be treated as DITA maps when opened in Oxygen. If the files match the pattern that is specified in **Treat as DITA Maps files matching** you will be prompted to open them in the **DITA Maps Manager** view.

The files that match the pattern that is specified in will be excluded from the following operations from ***Project** view* that will be applied on a set of files: **File/Replace in Files**, **Check Spelling in Files**, **Validate**.

SVN File Editors

Each type of file is associated with a type of editor which opens the files of that type for editing. The editor can be the built-in one specially provided for the file type (for example the internal XML editor, the internal XSLT editor, the internal XSL-FO editor, etc) or an external application installed on the computer, either the default system application associated with that file type in the operating system or other particular application specified by the path to its executable file. The list of all the associations file type - editor is displayed in the preferences panel **SVN File Editors** which is opened from menu **Options** > **Preferences** > **SVN File Editors** .

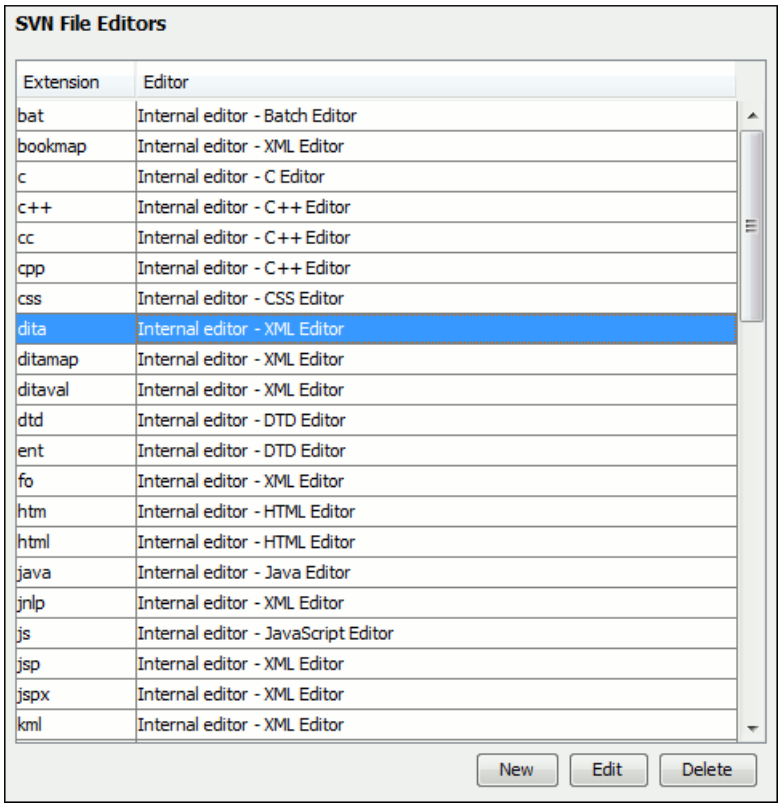


Figure 436: The SVN File Editors Preferences Panel

The **Edit** button or a double click on a table row opens a dialog for specifying the editor associated with the file type. The same dialog is displayed on opening a file from one of the Syncro SVN Client views.

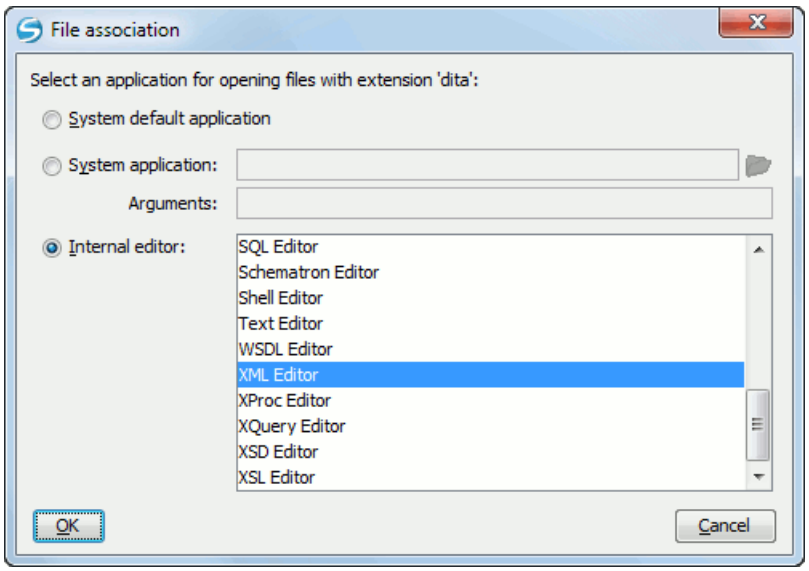


Figure 437: The Open With Dialog

In this dialog are offered three options for opening a file:

- **System default application** - Opens the selected file using the application that is associated with that file extension by default in the operating system.
- **System application** - Opens the selected file using an external application that you have to specify by the path of its executable file. Also, you can specify some arguments for the command line of that application, if they are needed. This option also works for directories, if you wish to choose a file browser other than the system default.
- **Internal editor** - Allows selecting an editor type from the built-in editors that Syncro SVN Client comes with. By default, this option is disabled when selecting directories.

If a file type is associated with an internal editor other than an XML editor type then the encoding set in *the preference **Encoding for non XML files*** is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the encoding of the file.

Custom Editor Variables

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.

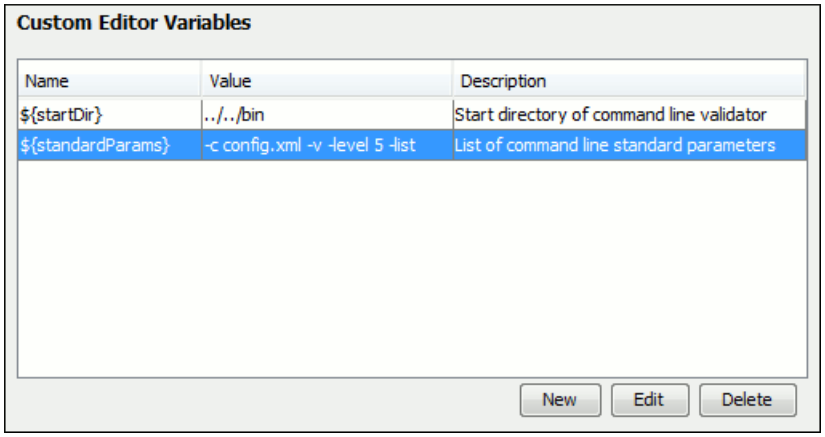


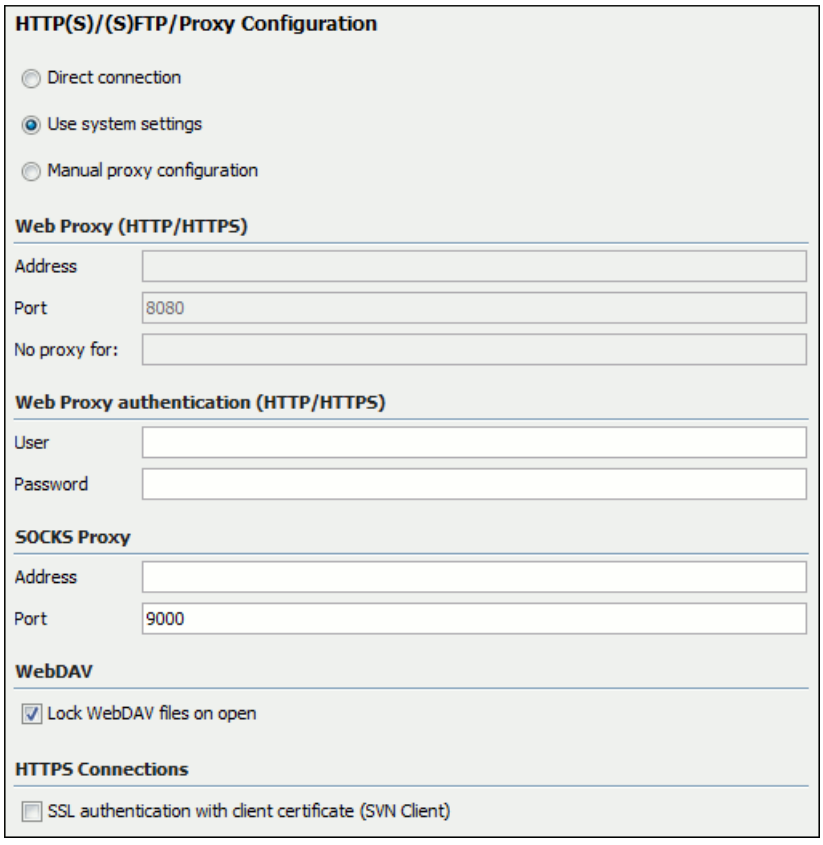
Figure 438: Custom Editor Variables

HTTP(S) / (S)FTP / Proxy Configuration

Some networks use proxy servers to provide Internet services to LAN clients. Clients behind the proxy may therefore, only connect to the Internet via the proxy service. If you are not sure whether your computer is required to use a proxy server to connect to the Internet or you don't know the proxy parameters, please consult your network administrator.

You can open the HTTP(S) / (S)FTP / Proxy Configuration panel from menu **Options** > **Preferences** > **HTTP(S) / (S)FTP / Proxy Configuration** .

The HTTP(S) / (S)FTP / Proxy Configuration Preferences Panel



Complete the dialog as follows:

- **Direct connection** - If checked, the HTTP(S) connections go directly to the target host without going through a proxy server.
- **Use system settings** - If checked, the HTTP(S) connections go through the proxy server set in the operating system. For example on Windows the proxy settings are the ones used by Internet Explorer.

  Attention: The system settings for the proxy cannot be read correctly from the operating system on some Linux systems. The system settings option should work properly on Gnome based Linux systems but it does not work on KDE based ones as *the Java virtual machine does not offer the necessary support yet*.

- **Manual proxy configuration** - If checked, the HTTP(S) connections go through the proxy server specified in the fields **Address** and **Port** of the section **Web Proxy (HTTP / HTTPS)**. Also this section specifies the hosts to which the connections must not go through a proxy server in the field **No proxy for**.
- **Web Proxy authentication (HTTP / HTTPS)** - In this section you set the user and password necessary for authentication with the proxy server. The user and password set here will be used both in case of manual proxy configuration and in case of system settings selected above.
- **SOCKS Proxy** - In this section you set host and port of a SOCKS proxy through which all the connections must pass. If the **Address** field is empty the connections will use no SOCKS proxy.
- **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.
- **SSL authentication with client certificate** - If checked and the SVN server accessed through the HTTPS protocol requires a digital certificate, the user is asked to specify a file path containing the digital certificate in the PKCS format for accessing that server.

Advanced HTTP Settings

The **Advanced HTTP Settings** preferences panel is opened from menu **Options** > **Preferences** > **HTTP(S) / (S)FTP / Proxy Configuration** > **Advanced HTTP Settings** and offers the following preferences:

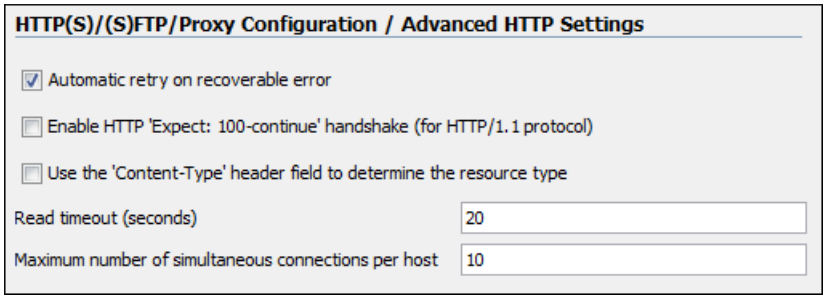


Figure 439: The Advanced HTTP Settings Preferences Panel

- **Automatic retry on recoverable error** - If enabled, if an HTTP error occurs when Oxygen communicates with a server via HTTP, for example sending / receiving a SOAP request / response to / from a Web services server, and the error is recoverable, Oxygen tries to send again the request to the server.
- **Enable HTTP 'Expect: 100-continue ' handshake for HTTP/1.1 protocol** - Activates *Expect: 100-Continue* handshake. The purpose of the *Expect: 100-Continue* handshake is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request headers) before the client sends the request body. The use of the *Expect: 100-continue* handshake can result in noticeable performance improvement when working with databases. The *Expect: 100-continue* handshake should be used with caution, as it may cause problems with HTTP servers and proxies that do not support the HTTP/1.1 protocol.
- **Use the 'Content-Type' header field to determine the content type** - When checked, Oxygen 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.

- **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.

  Tip: If the **Automatic retry on recoverable error** option is checked, the HTTP client tries to establish the connection twice so the timeout will be double the timeout specified here.

- **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.

(S)FTP

The **(S)FTP** preferences panel is opened from menu **Options** > **Preferences** > **HTTP(S) / (S)FTP / Proxy Configuration** > **(S)FTP** and offers the following preferences:

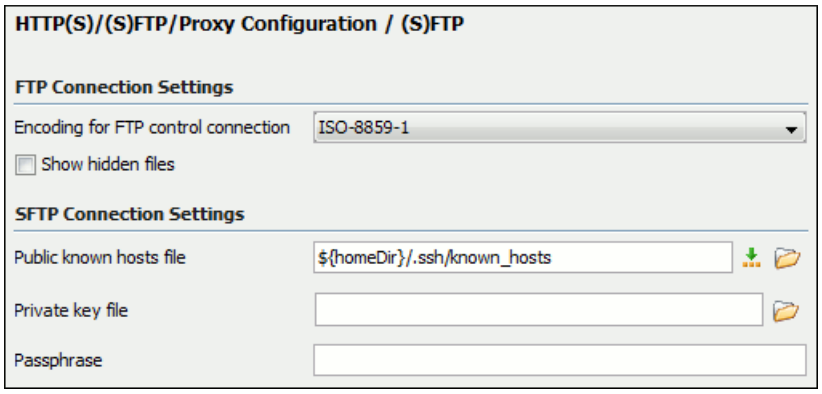


Figure 440: The (S)FTP Configuration Preferences Panel

- **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`.
- **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 will use it for communication. Otherwise it will use ISO-8859-1.
- **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol. The user / password method of authentication has precedence if it is used in *the Open URL dialog*.
- **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol. The user / password method of authentication has precedence if it is used in *the Open URL dialog*.

Certificates

In Oxygen there are provided two types of keystores for certificates used for digital signatures of XL documents: Java KeyStore (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A keystore file is protected by a password. A certificate keystore is configured in Oxygen in the **Certificates** preferences panel which is opened from menu **Options** > **Preferences** > **Certificates** . The parameters of a keystore are the following:

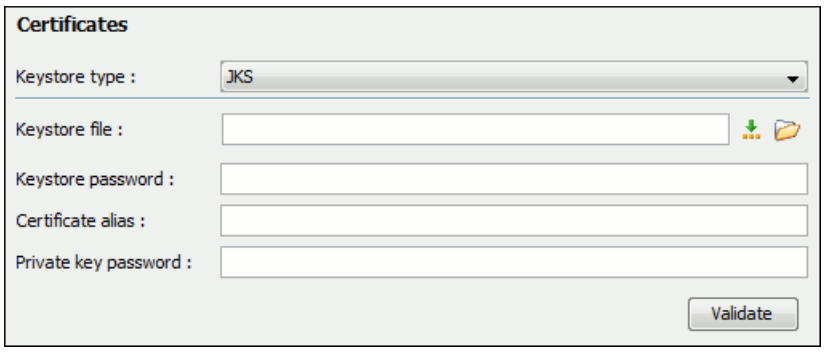


Figure 441: The Certificates Preferences Panel

- **Keystore type** - Represents the type of keystore to be used.
- **Keystore file** - Represents the location of the file to be imported.
- **Keystore password** - The password which is used to protect the privacy of the stored keys.
- **Certificate alias** - The alias to be used to store the key entry (the certificate and / or the private key) inside the keystore.
- **Private key password** - It is only necessary in case of JKS keystore. It represents the certificate's private key password.
- **Validate** - Pressing this button verifies the keystore configured with the above parameters and assures that the certificate is valid.

XML Structure Outline

The **XML Structure Outline** preferences panel is opened from menu **Options** > **Preferences** > **XML Structure Outline** and contains the following preferences:

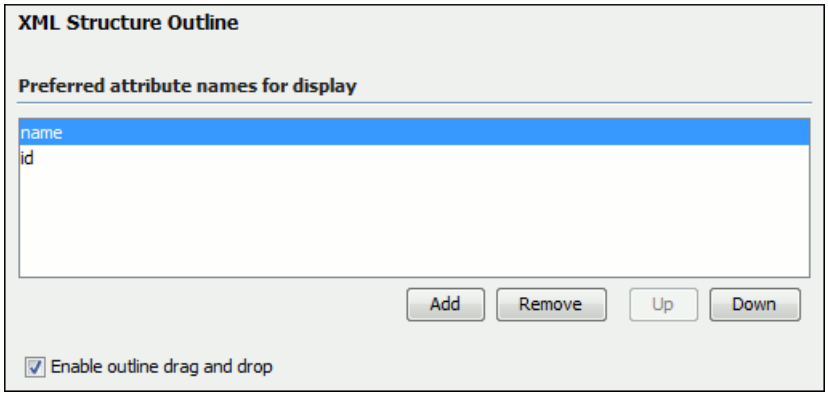


Figure 442: The XML Structure Outline Preferences Panel

- **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.

View

The **View** preferences panel is opened from menu **Options** > **Preferences** > **View** and contains the following preferences:

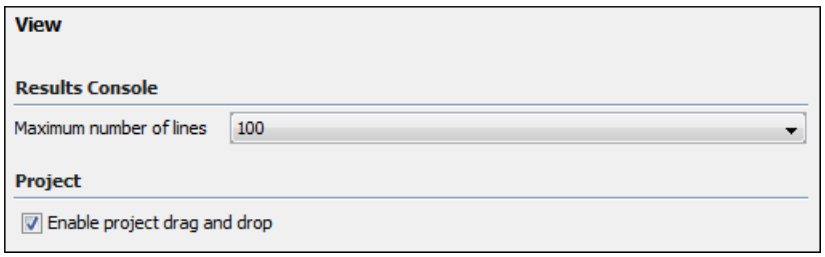


Figure 443: The View Preferences Panel

- **Maximum number of lines** - Sets the maximum number of lines of the output console where the output of the external tools is displayed.
- **Enable project drag and drop** - Enables the drag and drop support in **Project** view. It should be disabled only if there is a possibility to accidentally change the structure of the project by such drag and drop actions.

Messages

The **Messages** preferences panel is opened from menu **Options** > **Preferences** > **Messages** and allows disabling the following warning messages which may appear in the application:

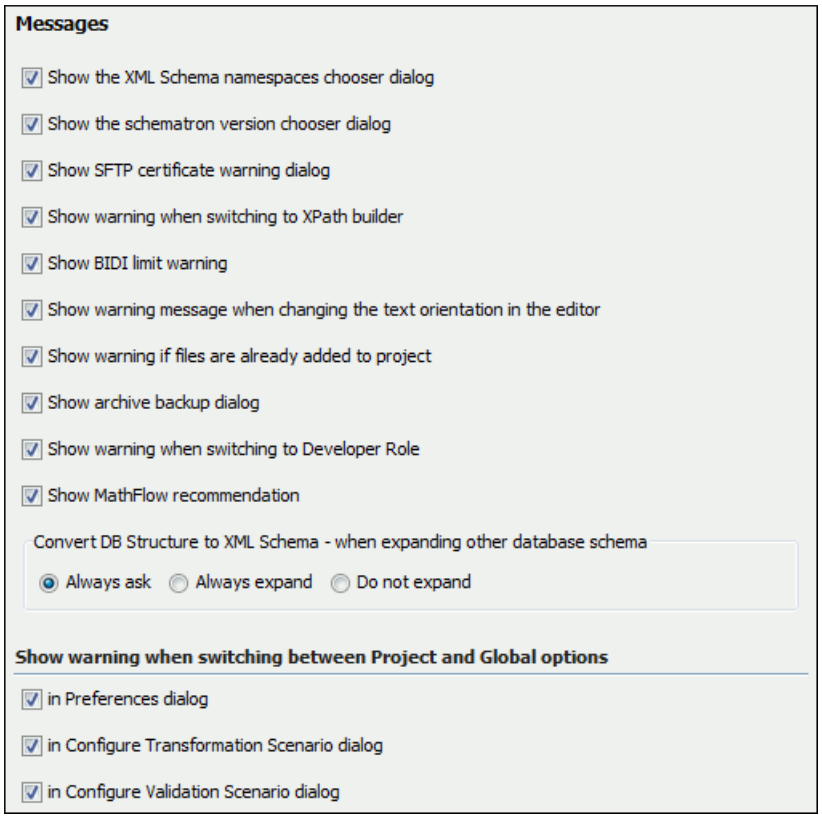


Figure 444: The Messages Preferences Panel

- **Show the XML Schema namespaces chooser dialog** - If checked, a namespace dialog appears when creating a new W3C XML Schema file.
- **Show the stylesheet version chooser dialog** - If checked, the XSLT version chooser dialog will be shown when creating a new XSLT stylesheet file.
- **Show the schematron version chooser dialog** - If checked, the Schematron version chooser dialog will be shown when creating a new schematron file.
- **Show SFTP Warning dialog** - If checked, a warning dialog will be shown each time when the authenticity of the SFTP server host cannot be established.

- **Show warning when switching to XPath builder** - If checked, a warning dialog will be shown when the XPath toolbar contains a long expression and the user is advised to use the **XPath Builder** view instead.
- **Show BIDI limit warning** - If checked, a warning dialog will be shown when the opened file which contains bidirectional characters is too large and bidirectional support is disabled.
- **Show warning if files are already added to project** - If checked, a dialog will be shown warning the user if he wants to add to project an already existing file.
- **Show archive backup dialog** - If checked, a dialog will be shown allowing the user different backup options before modifying an archive's content.
- **Show warning when switching to Developer Role** - If checked, a warning dialog will be displayed when choosing to switch to developer role in the *Document Type Association* panel.
- **Show MathFlow recommendation** - If checked, if the MathFlow component is not configured in Oxygen and you are trying to edit a MathML equation you will be notified that Oxygen can be easily configured to use the MathFlow editor.
- **Convert DB Structure to XML Schema - when expanding other database schema** - When tables from a database schema are selected in the **Select Database Table** dialog, after the **Convert DB Structure to XML Schema** is invoked, and another database schema is expanded, a confirmation is needed because the previous selection will be discarded. This option controls whether the user should always be asked about the next action, the other database schema will always be expanded without asking the user or it will never be expanded.
- **Show warning when switching between Project and Global options in Preferences dialog** - If checked, a warning dialog will be shown about uncommitted changes when switching between Project and Global options in a preferences panel.
- **Show warning when switching between Project and Global options in Configure Transformation Scenario dialog** - If checked, a warning dialog will be shown about uncommitted changes when switching between Project and Global options in the transformation scenarios edit dialog.
- **Show warning when switching between Project and Global options in Configure Validation Scenario dialog** - If checked, a warning dialog will be shown about uncommitted changes when switching between Project and Global options in the validation scenarios edit dialog.

Tree Editor

The **Tree Editor** preferences panel is opened from menu **Options** > **Preferences** > **Tree Editor** and contains the following preferences:

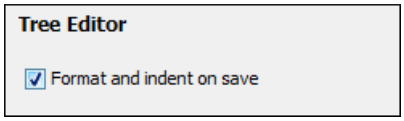


Figure 445: The Tree Editor Preferences Panel

- **Format and indent on save** - If selected, Oxygenwill run the action **Format and Indent** (pretty-print) on saving a document edited in the tree editor.

Sharing Preferences

By default all the user preferences are global and are stored in a subfolder of the user home folder. The user preferences can also be stored in the current project file that is loaded in the **Project** view. This allows you to share with your team a set of user preferences through an Oxygen project already configured and distributed to the team members.

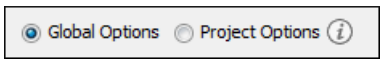


Figure 446: Controlling the Storage of the Preferences

For instance you may decide that the default schema associations and catalogs must be shared. In this case you simply click the radio button **Project Options** in the respective panels of the **Preferences** dialog, edit the preferences values and then save your Oxygen project in the **Project** view. If you want to drop later a preferences panel from being stored into the project, you select the radio button **Global Options** and press the OK button in the **Preferences** dialog.

Automatically Importing the Preferences from the Other Distribution

If you want to use the settings from Oxygenstandalone distribution in the Eclipse plugin one just delete the file with the Eclipse plugin preferences, that is `[user-home-folder]/Application Data/com.oxygenxml/oxyOptionsEc12.2.xml` on Windows / `[user-home-folder]/.com.oxygenxml/oxyOptionsEc12.2.xml` on Linux, start Eclipse and the Oxygen standalone preferences will be automatically imported in Eclipse. The same for importing the Eclipse plugin settings in Oxygen standalone: delete the file `[user-home-folder]/com.oxygenxml/oxyOptionsSa12.2.xml` on Windows / `[user-home-folder]/.com.oxygenxml/oxyOptionsSa12.2.xml` on Linux, start the Oxygen standalone distribution and the Eclipse settings will be automatically imported in Oxygen standalone.

Reset Global Options

To reset all global preferences to their default values you have to go to menu **Options** > **Reset Global Options** > **Reset Global Options** . . The project level preferences are not changed by this action. The list of transformation scenarios will be reset to the default scenarios.

Scenarios Management

You can import, export and reset the global scenarios for transformation and validation with the following actions:

- The action **Options** > **Import Global Transformation Scenarios** loads a set of transformation scenarios from a properties file that was created with the action **Export Global Transformation Scenarios**.
- The action **Options** > **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**.
- The action **Options** > **Import Global Validation Scenarios** loads a set of validation scenarios from a properties file that was created with the action **Export Global Validation Scenarios**.
- The action **Options** > **Export Global Validation Scenarios** stores all the global (not project-level) validation scenarios in a separate properties file.

The options of **Export Global Transformation Scenarios** and **Export Global Validation Scenarios** is used to store all the scenarios in a separate file which is a properties file. In this file will also be saved the associations between document URLs and scenarios. You can load the saved scenarios using the actions **Import Global Transformation Scenarios** and **Import Global Validation Scenarios**. All the imported scenarios will have added to the name the word **import** to distinguish the existing scenarios and the imported ones.

Editor Variables

An editor variable is a shorthand notation for a file path or folder path. It is used in the definition of a command (the input URL of a transformation, the output file path of a transformation, the command line of an external tool, etc.) 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.

The following editor variables can be used in Oxygen commands of external engines or other external tools, in transformation scenarios and in validation scenarios:

- **${oxygenHome}** - Oxygen installation folder as URL.
- **${oxygenInstallDir}** - Oxygen installation folder as file path.
- **${frameworks}** - The path (as URL) of the `frameworks` subfolder of the Oxygen install folder.
- **${frameworksDir}** - The path (as file path) of the `frameworks` subfolder of the Oxygen installation folder.
- **${home}** - The path (as URL) of the user home folder.
- **${homeDir}** - The path (as file path) of the user home folder.
- **${pdu}** - Current project folder as URL.
- **${pd}** - Current project folder as file path.
- **${pn}** - Current project name.
- **${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.
- **${cf}** - Current file as file path, that is the absolute file path of the current edited document.
- **${currentFileURL}** - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- **${ps}** - Path separator, that is the separator which can be used on the current platform (Windows, Mac OS X, Linux) between library files specified in the class path.
- **${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.
- **${caret}** - The position where the caret is inserted. This variable can be used in a *code template* or in a *selection plugin*.
- **${selection}** - The text content of the current selection in the editor panel. This variable can be used in a *code template* or in a *selection plugin*.

Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of an *external tool* or of a *custom validator*, the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

An editor variable can be created also from a Java system property. For example the Java system property `var.name` can be inserted in any expression where built-in editor variables like **${currentFileURL}** are allowed: just type **${system(var.name)}** in the field where you need that Java system property.

An editor variable can be created also from an environment variable of the operating system. For example the environment variable `VAR_NAME` can be inserted in any expression where built-in editor variables like **${currentFileURL}** are allowed: just type **${env(VAR_NAME)}** where you need the environment variable.

The current date can be inserted at cursor location with the custom variable **${date(yyyy-MM-dd)}**. The date format is: the year with 4 digits, the month with 2 letters, the day with 2 letters.

The custom editor variables are *configured in Preferences*.

Configure Toolbars

You can configure the toolbars that appear in the current editor page (Author mode, Text mode, Grid mode, Schema Design mode) using the **Configure toolbars** dialog that is opened from menu **Window** > **Configure toolbars ...** .

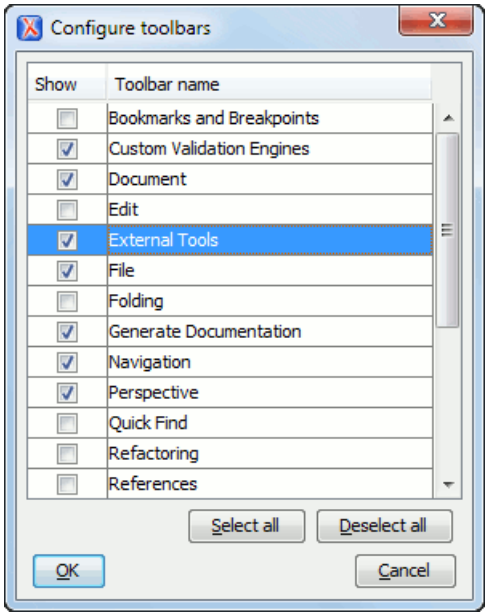


Figure 447: Configure Toolbars Dialog

Chapter 25

Common Problems

Topics:

- *XML Document Opened After a Long Time*
- *Out Of Memory Error When I Open Large Documents*
- *Special Characters Are Replaced With a Square in Editor*
- *XSLT Debugger Is Very Slow*
- *The Scroll Function of my Notebook's Trackpad is Not Working*
- *NullPointerException at Startup on Windows XP*
- *Crash at Startup on Windows with an Error Message About a File nvoglv32.dll*
- *Oxygen Crashed on My Mac OS X Computer*
- *Wrong Highlights of Matched Words in a Search in User Manual*
- *Keyboard Shortcuts Do Not Work*
- *After Installing Oxygen I Cannot Open XML Files in Internet Explorer Anymore*
- *I Cannot Associate Oxygen With a File Type on My Windows Computer*
- *The Files Are Opened in Split Panels When I Restart the Oxygen Application*
- *Grey Window on Linux With the Compiz / Beryl Window Manager*
- *Drag and Drop Without Initial Selection Does Not Work*
- *Set Specific JVM Version on Mac OS X*
- *Segmentation Fault Error on Mac OS X*

This chapter presents common problems that may appear when running the application and the solutions for these problems.

- *Damaged File Associations on Mac OS X*
- *I Cannot Set XML Files to Open With Oxygen*
- *I Cannot Connect to SVN Repository From Repositories View*
- *Problem Report Submitted on the Technical Support Form*
- *Cannot cancel a system shutdown when there is at least one modified document open in Oxygen*

XML Document Opened After a Long Time

Oxygen opens an XML document after a long time. Why does it happen?

Usually the time necessary for opening an XML document is long when the whole content of your document is on a single line or the document size is very large. If the content is on a single line please enable the **Format and indent the document on open** preference from menu ***Options*** > ***Preferences*** > ***Editor*** > ***Format*** before opening the document. If the document is very large (above 30 MB) you should make sure that the minimum limit of document size that enables the support for editing large documents (the value of *the option* ***Optimize loading in the Text page for files over***) is less than the size of your document. If that fails and you get an Out Of Memory error (**OutOfMemoryError**) you should *increase the memory available to Oxygen.*

Out Of Memory Error When I Open Large Documents

I am trying to open a file larger than 100 MB for editing in Oxygen but it keeps telling me it runs out of memory (**OutOfMemoryError**). What can I do?

You should make sure that the minimum limit of document size that enables the support for editing large documents (the value of *the option* ***Optimize loading in the Text page for files over***) is less than the size of your document. That will enable the optimized support for large documents. If that fails and you still get an Out Of Memory error you should *increase the memory available to Oxygen.*

Special Characters Are Replaced With a Square in Editor

My file was created with other application and it contains special characters like é, ©, ®, etc. Why does Oxygen display a square for these characters when I open the file in Oxygen ?

You must set a font able to render the special characters in the ***Font*** *preferences*. If it is a text file you must set also *the encoding used for non XML files.* If you want to set a font which is installed on your computer but that font is not accessible in the **Font** preferences that means the Java virtual machine is not able to load the system fonts, probably because it is not a True Type font. It is a problem of the Java virtual machine and a possible solution is to copy the font file in the `[JVM-home-folder]/lib/fonts` folder. `[JVM-home-folder]` is the value of the property *java.home* which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help** > **About** .

XSLT Debugger Is Very Slow

When I run a transformation in the **XSLT Debugger** perspective it is very slow. Can I increase the speed?

If the transformation produces HTML or XHTML output you should *disable rendering of output in the XHTML output view* during the transformation process. In order to view the XHTML output result you can run the transformation in the **Editor** perspective with *the option* ***Open in browser*** enabled or run it in the **XSLT Debugger** perspective, save the text output area to a file and use a browser application for viewing it, for example Firefox or Internet Explorer.

The Scroll Function of my Notebook's Trackpad is Not Working

I got a new notebook (Lenovo Thinkpad™ with Windows) and noticed that the scroll function of my trackpad is not working in Oxygen XML Editor .

It is a problem of the Synaptics™ trackpads which can be fixed by adding the following lines to the `C:\Program Files\Synaptics\SynTP\TP4table.dat` file:

```
*,*,oxygen.exe,*,*,*,WheelStd,1,9
*,*,author.exe,*,*,*,WheelStd,1,9
*,*,syncroSVNClient.exe,*,*,*,WheelStd,1,9
*,*,diffDirs.exe,*,*,*,WheelStd,1,9
*,*,diffFiles.exe,*,*,*,WheelStd,1,9
```

`NullPointerException` at Startup on Windows XP

When I start Oxygen on Windows XP I get the following error. What can I do?

```
Cannot start Oxygen.
Due to:java.lang.NullPointerException
java.lang.NullPointerException
at com.sun.java.swing.plaf.windows.XPStyle.getString(Unknown Source)
at com.sun.java.swing.plaf.windows.XPStyle.getString(Unknown Source)
at com.sun.java.swing.plaf.windows.XPStyle.getDimension(Unknown Source)
at com.sun.java.swing.plaf.windows.WindowsProgressBarUI.
getPreferredInnerHorizontal(Unknown Source)
```

The error is caused by a *a bug in the Java runtime from Sun Microsystems*. You can avoid it by setting the Java system property *com.oxygenxml.no.xp.theme* to the value *true* in the startup script, that is *adding the startup parameter -Dcom.oxygenxml.no.xp.theme=true*. If you start Oxygen with the oxygen.bat script just add the parameter-Dcom.oxygenxml.no.xp.theme=true to the `java` command in the script. If you start Oxygen from the Start menu shortcut add the same parameter on a new line in the file `[oXygen-install-folder]\oxygen.vmoptions` .

Crash at Startup on Windows with an Error Message About a File `nvoglv32.dll`

I try to start Oxygen on Windows but it crashed with an error message about “Fault Module Name: nvoglv32.dll”. What is the problem?

It is an OpenGL driver issue that can be avoided by creating an empty file called `opengl32.dll` in the Oxygen install folder (if you start Oxygen with the shortcut created by the installer on the Start menu or on Desktop) or in the subfolder `bin` of the home folder of the Java virtual machine that runs Oxygen (if you start Oxygen with the `oxygen.bat` script). The home folder of the Java virtual machine that runs Oxygen is the value of the property *java.home* which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help** > **About** .

Oxygen Crashed on My Mac OS X Computer

Oxygen crashed the Apple Java virtual machine/Oxygen could not start up on my Mac OS X computer due to a JVM crash. What can I do?

Usually it is an incompatibility between the Apple JVM and a native library of the host system. More details are available in the crash log file generated by OS X and reported in the crash error message.

Wrong Highlights of Matched Words in a Search in User Manual

When I do a keyword search in the User Manual that comes with the Oxygen application the search highlights the wrong word in the text. Sometimes the highlighted word is several words after the matched word. What can I do to get correct highlights?

This does not happen when Oxygen runs with a built-in Java virtual machine, that is a JVM that was installed by Oxygen in a subfolder of the installation folder, for example on Windows and Linux when installing Oxygen with the installation wizard specific for that platform. When Oxygen runs from an All Platforms installation it uses whatever JVM was found on the host system which may be incompatible with the JavaHelp indexer used for creating the built-in User Manual. Such a JVM may offset the highlight of the matched word with several characters, usually to the right of the matched word. In order to see the highlight exactly on the matched word it is recommended to install the application with the specific installation wizard for your platform (available only for Windows and Linux).

Keyboard Shortcuts Do Not Work

The keyboard shortcuts listed in **Options** > **Preferences** > **Menu Shortcut Keys** do not work. What can I do?

Usually this happens when a special keyboard layout is set in the operating system which generates other characters than the usual ones for the keys of a standard keyboard. For example if you set the extended Greek layout for your keyboard you should return to the default Greek layout or to the English one. Otherwise the Java virtual machine that runs the application will receive other key codes than the usual ones for a standard keyboard.

After Installing Oxygen I Cannot Open XML Files in Internet Explorer Anymore

Before installing Oxygen I had no problems opening XML files in Internet Explorer. Now when I try to open an XML file in Internet Explorer it opens the file in Oxygen. How can I load XML files in Internet Explorer again?

XML files are opened in Oxygen because Internet Explorer uses the Windows system file associations for opening files and you associated XML files with Oxygen in the installer panel called **File Associations** therefore Internet Explorer opens XML files with the associated Windows application.

By default the association with XML files is disabled in the Oxygen installer panel called **File Associations**. When you enabled it the installer displayed a warning message about the effect that you experience right now.

For opening XML files in Internet Explorer again you have to set Internet Explorer as the default system application for XML files, for example by right-clicking on an XML file in Windows Explorer, selecting **Open With** > **Choose Program** , selecting IE in the list of applications and selecting the checkbox **Always use the selected program**. Also you have to run the following command from a command line:

```
wscript revert.vbs
```

where `revert.vbs` is a text file with the following content:

```
function revert()
  Set objShell = CreateObject("WScript.Shell")
  objShell.RegWrite "HKCR\.xml\", "xmlfile", "REG_SZ"
  objShell.RegWrite "HKCR\.xml\Content Type", "text/xml", "REG_SZ"
end function

revert()
```

I Cannot Associate Oxygen With a File Type on My Windows Computer

I cannot associate the Oxygen application with a file type on my Windows computer by right clicking on a file in Windows Explorer, selecting **Open With** > **Choose Program** and browsing to the file `oxygen.exe`. When I select the file `oxygen.exe` in the Windows file browser dialog the Oxygen application is not added to the list of applications in the **Open With** dialog. What can I do?

The problem is due to some garbage Windows registry entries remained from versions of Oxygen older than version 9.0. Please uninstall all your installed versions of Oxygen and run a registry cleaner application for cleaning these older entries. After that just reinstall your current version of Oxygen and try again to create the file association.

The Files Are Opened in Split Panels When I Restart the Oxygen Application

When I close the Oxygen application with multiple files open and then restart it, every file opens in a split panel of the editing area instead of a tab sharing with the other opened files the same editing area which organizes the editors in a tabbed pane. I want to have the files arranged as a tabbed pane as they used to be arranged before closing the Oxygen application.

This happens sometimes when several files are opened automatically on startup. It is a problem of the JIDE docking views library used in Oxygen for docking and floating views. The workaround is to run the action **Window** > **Reset Layout** . If you have a specific layout of the Oxygen views which you want to preserve when running this action you should set your layout with the option **Use fixed layout** that is available from menu **Options** > **Preferences** > **Perspectives Layout** .

Grey Window on Linux With the Compiz / Beryl Window Manager

I try to run Oxygen on Linux with the Compiz / Beryl window manager but I get only a grey window which does not respond to user actions. Sometimes the Oxygen window responds to user actions but after opening and closing an Oxygen dialog or after resizing the Oxygen window or a view of the Oxygen window the content of this window becomes grey and it does not respond to user actions. What is wrong?

Sun Microsystems' Java virtual machine *does not support the Compiz window manager and the Beryl one very well*. It is expected that better support for Compiz / Beryl will be added in future versions of their Java virtual machine. You should turn off the special effects of the Compiz / Beryl window manager before starting the Oxygen application or switch to other window manager.

Drag and Drop Without Initial Selection Does Not Work

When I try to drag with the mouse an unselected file from the **Project** view or the **DITA Maps Manager** view, the drag never starts, it only selects the resource. I need to drag the resource again after it becomes selected. As a result any drag and drop without initial selection becomes a two step operation. How can I fix this?

This is *a bug* present in JVM versions prior to 1.5.0_09. This issue is fixed in 1.5.0_09 and newer versions (including 1.6). See *the installation instructions* for setting a specific JVM version for running the Oxygen application.

Set Specific JVM Version on Mac OS X

How do I configure Oxygen to run with the version X of the Apple Java virtual machine on my Mac OS X computer?

Oxygen uses the first JVM from the list of preferred JVM versions set on your Mac computer that has a version number 1.5.0 or higher. You can move your desired JVM version up in the preferred list by dragging it with the mouse on a higher position in the list of JVMs available in the **Java Preferences** panel that is opened from **Applications** > **Utilities** > **Java** > **Java Preferences** .

Segmentation Fault Error on Mac OS X

On my Mac OS X machine the application gives a *Segmentation fault* error when I double-click on the application icon. Sometimes it gives no error but it does not start. What is the problem?

Please make sure you have the latest Java update from the Apple website installed on your Mac OS X computer. If installing the latest Java update doesn't solve the problem please copy the file `JavaApplicationStub` from the `/System/Frameworks/JavaVM.framework` folder to the `Oxygen.app/Contents/MacOS` folder. For browsing the `.app` folder you have to **(CMD+click)** on the Oxygen icon and select **Show Package Contents**.

Damaged File Associations on Mac OS X

After upgrading OS X to version 10.4.x / Oxygen to version 6.x Oxygen is not associated anymore to the file types XML, XSL, XSD, etc. This worked in the previous version of Oxygen. How can I create the file associations again?

The upgrade damaged the file associations in the LaunchService Database on your Mac OS X machine. Please rebuild the LaunchService Database with the following procedure. This will reset all file associations and will rescan the entire file system searching for applications that declare file associations and collecting them in a database used by Finder.

1. Find all the Oxygen installations on your hard drive.
2. Delete them by dragging them to the Trash.
3. Clear the Trash.
4. Unpack the Oxygen installation kit on your desktop.
5. Copy the contents of the archive into the folder `/ Applications / Oxygen`.
6. Run the following command in a Terminal:

```
/System/Library/Frameworks/CoreServices.framework/Frameworks/
LaunchServices.framework/Support/lsregister -kill -r -domain local -domain
system -domain user -dump
```

7. Restart Finder with the following command:

```
killall Finder
```

8. Create a XML or XSD file on your desktop.
   It should take the Oxygen icon.
9. Double click the file.
10. Accepting the confirmation dialog.

Oxygen will start up and the file associations work correctly.

I Cannot Set XML Files to Open With Oxygen

After upgrading OS X to version 10.4.1 Tiger I am not able to set XML files to open with Oxygen when I click **Change All** in the **Get Info** dialog. This worked in OS X 10.3.x.

On Mac OS X Tiger you must add an entry to the `Info.plist` file of the Oxygen bundle. Tiger was released after Oxygen version 6.0 so this change could not be included in the Oxygen 6.0 release. Please close Oxygen, press **(Cmd+click)** on the Oxygen icon, select **Show Package Contents**, go to **Contents**, edit the `Info.plist` file, add the entry:

```
    <key>CFBundleIdentifier</key>
    <string>ro.sync.exml.Oxygen</string>
```

and restart Oxygen. Now you can select **Change All** in the **Get Info** dialog to make the association.

I Cannot Connect to SVN Repository From Repositories View

I cannot connect to a SVN repository from the **Repositories** view of SVN Client. How can I find more details about the error?

First check that you entered the correct URL of the repository in the **Repositories** view. Also check that a SVN server is running on the server machine specified in the repository URL and is accepting connections from SVN clients. You can check that the SVN server accepts connections with the command line SVN client from CollabNet.

If you try to access the repository with a `svn+ssh` URL also check that a SSH server is running on port 22 on the server machine specified in the URL.

If the above conditions are verified and you cannot connect to the SVN repository please generate a logging file on your computer and send the logging file to support@oxygenxml.com . For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error, close the application and send the file `logging.log` generated in the install directory to support@oxygenxml.com .

Problem Report Submitted on the Technical Support Form

What details should I add to my problem report that I enter on the Technical Support online form of the product website?

For problems like server connection error, unexpected delay while editing a document, a crash of the application, etc for which the usual details requested on the Technical Support online form are not enough you should generate a log file and attach it to the problem report. In case of a crash you should also attach the crash report file generated by your operating system. For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error and close the application. The log file is called `logging.log` and is located in the install folder.

Cannot cancel a system shutdown when there is at least one modified document open in Oxygen

If I try to shutdown my Win XP when there is at least one modified document open in Oxygen, I am prompted to cancel the shutdown or force quit the application. If I choose **Cancel**, the system shuts down anyway. Why is that?

This problem was reported on Windows XP systems only. The known workaround is to start Oxygen using the `oxygen.bat` script.