Oxygen XML Developer Eclipse Plugin 17.0
Notice

Copyright

Oxygen XML Developer plugin User Manual
Syncro Soft SRL.

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

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

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

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

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

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

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

Downloading documents. For the most current versions of documentation, see the Oxygen XML Developer plugin website.

Contact Syncro Soft SRL. Syncro Soft SRL provides telephone numbers and e-mail addresses for you to report problems or to ask questions about your product, see the Oxygen XML Developer plugin website.
## Contents

**Chapter 1: Introduction**..................................................................................................................15

**Chapter 2: Installation**..................................................................................................................19

- Installation Options.........................................................................................................................20
- Windows Installation.........................................................................................................................20
- Mac OS X Installation.........................................................................................................................21
- Linux Installation...............................................................................................................................23
- Site-wide deployment.........................................................................................................................24
- Licensing...........................................................................................................................................24
- Setting up a License Server..............................................................................................................27
  - Setting up a Floating License Server Running as a Standalone Process Using a Platform-independent Distribution........................................................................................................31
- Transferring or Releasing a License..................................................................................................32
- Upgrading.........................................................................................................................................33
- Uninstalling.......................................................................................................................................34

**Chapter 3: Perspectives**................................................................................................................35

- Perspectives.......................................................................................................................................36
  - Oxygen XML Developer plugin XML Perspective ........................................................................36
  - XSLT Debugger Perspective............................................................................................................40
  - XQuery Debugger Perspective........................................................................................................40
  - Oxygen XML Developer plugin Database Perspective.............................................................41

**Chapter 4: Editing Modes**............................................................................................................43

- Text Editing Mode............................................................................................................................44
  - Finding and Replacing Text in the Current File............................................................................44
- Grid Editing Mode.............................................................................................................................45
  - Layouts: Grid and Tree.......................................................................................................................46
  - Grid Move Navigation.......................................................................................................................47
  - Specific Grid Actions.........................................................................................................................48
  - Drag and Drop in the Grid Editor.....................................................................................................49
  - Copy and Paste in the Grid Editor....................................................................................................49
  - Bidirectional Text Support in Grid Mode.......................................................................................51

**Chapter 5: Editing Documents**......................................................................................................53

- Working with Unicode.......................................................................................................................54
<table>
<thead>
<tr>
<th>Editing XML Schemas</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema Diagram Editing Mode</td>
<td>155</td>
</tr>
<tr>
<td>XML Schema Text Editing Mode</td>
<td>188</td>
</tr>
<tr>
<td>Editing XML Schema in the Master Files Context</td>
<td>189</td>
</tr>
<tr>
<td>Searching and Refactoring Actions in XML Schemas</td>
<td>189</td>
</tr>
<tr>
<td>Component Dependencies View</td>
<td>190</td>
</tr>
<tr>
<td>XML Schema Quick Assist Support</td>
<td>192</td>
</tr>
<tr>
<td>XML Schema Resource Hierarchy / Dependencies View</td>
<td>192</td>
</tr>
<tr>
<td>Generating Documentation for an XML Schema</td>
<td>195</td>
</tr>
<tr>
<td>Flatten an XML Schema</td>
<td>202</td>
</tr>
<tr>
<td>Generate Sample XML Files</td>
<td>204</td>
</tr>
<tr>
<td>XML Schema Regular Expressions Builder</td>
<td>208</td>
</tr>
<tr>
<td>Create an XML Schema From a Relational Database Table</td>
<td>209</td>
</tr>
<tr>
<td>XML Schema 1.1</td>
<td>209</td>
</tr>
<tr>
<td>Setting the XML Schema Version</td>
<td>210</td>
</tr>
<tr>
<td>Editing XQuery Documents</td>
<td>211</td>
</tr>
<tr>
<td>XQuery Outline View</td>
<td>211</td>
</tr>
<tr>
<td>Folding in XQuery Documents</td>
<td>212</td>
</tr>
<tr>
<td>Formatting and Indenting XQuery Documents</td>
<td>213</td>
</tr>
<tr>
<td>Generating HTML Documentation for an XQuery Document</td>
<td>213</td>
</tr>
<tr>
<td>Editing WSDL Documents</td>
<td>214</td>
</tr>
<tr>
<td>WSDL Outline View</td>
<td>214</td>
</tr>
<tr>
<td>Content Completion in WSDL Documents</td>
<td>217</td>
</tr>
<tr>
<td>Editing WSDL Documents in the Master Files Context</td>
<td>218</td>
</tr>
<tr>
<td>Searching and Refactoring Operations in WSDL Documents</td>
<td>219</td>
</tr>
<tr>
<td>Searching and Refactoring Operations Scope in WSDL Documents</td>
<td>220</td>
</tr>
<tr>
<td>WSDL Resource Hierarchy/Dependencies View in WSDL Documents</td>
<td>220</td>
</tr>
<tr>
<td>Component Dependencies View in WSDL Documents</td>
<td>223</td>
</tr>
<tr>
<td>Highlight Component Occurrences in WSDL Documents</td>
<td>224</td>
</tr>
<tr>
<td>Quick Assist Support in WSDL Documents</td>
<td>225</td>
</tr>
<tr>
<td>Generating Documentation for WSDL Documents</td>
<td>225</td>
</tr>
<tr>
<td>WSDL SOAP Analyzer</td>
<td>229</td>
</tr>
<tr>
<td>Editing CSS Stylesheets</td>
<td>232</td>
</tr>
<tr>
<td>Validating CSS Stylesheets</td>
<td>232</td>
</tr>
<tr>
<td>Content Completion in CSS Stylesheets</td>
<td>232</td>
</tr>
<tr>
<td>CSS Outline View</td>
<td>233</td>
</tr>
<tr>
<td>Folding in CSS Stylesheets</td>
<td>233</td>
</tr>
<tr>
<td>Formatting and Indenting CSS Stylesheets (Pretty Print)</td>
<td>234</td>
</tr>
<tr>
<td>Minifying CSS Stylesheets</td>
<td>234</td>
</tr>
<tr>
<td>Other CSS Editing Actions</td>
<td>234</td>
</tr>
<tr>
<td>Editing LESS CSS Stylesheets</td>
<td>234</td>
</tr>
<tr>
<td>Validating LESS Stylesheets</td>
<td>235</td>
</tr>
<tr>
<td>Content Completion in LESS Stylesheets</td>
<td>235</td>
</tr>
<tr>
<td>Compiling LESS Stylesheets to CSS</td>
<td>235</td>
</tr>
</tbody>
</table>
Chapter 6: Predefined Document Types.................................................................279

Document Type...........................................................................................................281
- The DocBook 4 Document Type..............................................................................281
- The DocBook 5 Document Type..............................................................................288
- The DITA Topics Document Type............................................................................294
- The DITA Map Document Type..............................................................................295
- The XHTML Document Type..................................................................................302
- The TEI ODD Document Type..................................................................................303
- The TEI P4 Document Type....................................................................................303
- The TEI P5 Document Type....................................................................................304
- The JATS Document Type.......................................................................................305
- The EPUB Document Type......................................................................................305
- The DocBook Targetset Document Type...............................................................306

Chapter 7: Transforming Documents........................................................................307

Transformation Scenarios..............................................................................................308
- Defining a New Transformation Scenario...............................................................308
- Configure Transformation Scenario(s) Dialog Box.................................................334
- Duplicating a Transformation Scenario.................................................................336
- Editing a Transformation Scenario........................................................................336
- Apply Batch Transformations..................................................................................336
- Built-in Transformation Scenarios...........................................................................337

Transformation Scenarios View....................................................................................337
- Debugging PDF Transformations............................................................................339
- XSLT Processors......................................................................................................340
- XSL-FO Processors...................................................................................................342

Chapter 8: Querying Documents.................................................................................367

Running XPath Expressions..........................................................................................368
- What is XPath..........................................................................................................368
- The XPath/XQuery Builder View.............................................................................368
Chapter 9: Debugging XSLT Stylesheets and XQuery Documents........381
  Overview.................................................................382
  Layout.................................................................................382
    Control Toolbar............................................................383
    Information View..........................................................385
    Multiple Output Documents in XSLT 2.0 and XSLT 3.0..............394
  Working with the XSLT / XQuery Debugger..........................394
    Steps in a Typical Debug Process..................................394
    Using Breakpoints.......................................................395
    Determining What XSLT / XQuery Expression Generated Particular Output..............................................395
  Debugging Java Extensions..............................................397
  Supported Processors for XSLT / XQuery Debugging...............397

Chapter 10: Performance Profiling of XSLT Stylesheets and XQuery Documents........................................399
  Overview..............................................................................400
  Viewing Profiling Information...........................................400
    Invocation Tree View....................................................400
    Hotspots View.............................................................400
  Working with XSLT/XQuery Profiler....................................401

Chapter 11: Working with Archives........................................403
  Browsing and Modifying Archive Structure........................404
  Working with EPUB.........................................................405
    Create an EPUB..........................................................406
    Publish to EPUB..........................................................407
  Editing Files From Archives..............................................407

Chapter 12: Working with Databases........................................409
Chapter 13: Importing Data...........................................................................................................457

Introduction........................................................................................................................................458
Import from Database..........................................................................................................................458
Import Table Content as XML Document..........................................................................................458
Convert Table Structure to XML Schema.........................................................................................461
Import from MS Excel Files............................................................................................................461
Import from MS Excel 2007-2010 (.xlsx)..........................................................................................463
Import from HTML Files...................................................................................................................463
Import from Text Files.......................................................................................................................463
Import Content Dynamically............................................................................................................464

Chapter 14: Content Management System (CMS) Integration.........................................................467

Integration with Documentum (CMS) (deprecated).........................................................................468
Configure Connection to Documentum Server.................................................................................468
Documentum (CMS) Actions in the Data Source Explorer View.....................................................469
Transformations on DITA Content from Documentum (CMS).........................................................473
Integration with Microsoft SharePoint...............................................................................................473
How to Configure a SharePoint Connection......................................................................................473
SharePoint Connection Actions.......................................................................................................474

Chapter 15: Tools...............................................................................................................................477
Chapter 16: Configuring Oxygen XML Developer plugin

Preferences...................................................................................................................486
  Oxygen XML Developer plugin License...............................................................486
  Archive Preferences..............................................................................................487
  CSS Validator Preferences....................................................................................487
  Custom Editor Variables Preferences.................................................................487
  Data Sources Preferences....................................................................................488
  DITA Preferences..................................................................................................492
  Document Type Association Preferences.........................................................492
  Editor Preferences..................................................................................................502
  Fonts Preferences..................................................................................................521
  Network Connection Settings Preferences.........................................................521
  Scenarios Management Preferences.................................................................523
  View Preferences.................................................................................................523
  XML Preferences..................................................................................................523
  XML Structure Outline Preferences....................................................................542
Importing / Exporting Global Options....................................................................542
Reset Global Options..............................................................................................543
Customizing Default Options.................................................................................543
Scenarios Management..........................................................................................544
Editor Variables......................................................................................................544
  Custom Editor Variables......................................................................................548
Localization of the User Interface..........................................................................548

Chapter 17: Common Problems

Performance Problems............................................................................................552
  External Processes...............................................................................................552
Common Problems and Solutions............................................................................552
  Oxygen XML Developer plugin Takes Several Minutes to Start on Mac............552
  XSLT Debugger Is Very Slow...............................................................................552
  Syntax Highlight Not Available in Eclipse Plugin..............................................552
  Damaged File Associations on OS X.....................................................................553
  Problem Report Submitted on the Technical Support Form...............................553
  Signature verification failed error on open or edit a resource from Documentum...553
  Compatibility Issue Between Java and Certain Graphics Card Drivers................554
An Image Appears Stretched Out in the PDF Output..............................................................554
The DITA PDF Transformation Fails.........................................................................................555
The DITA to CHM Transformation Fails..................................................................................555
DITA Map ANT Transformation Because it Cannot Connect to External Location..............556
Topic References outside the main DITA Map folder...............................................................556
The PDF Processing Fails to Use the DITA OT and Apache FOP..........................................556
The TocJS Transformation Doesn't Generate All Files for a Tree-Like TOC......................557
Navigation to the web page was canceled when viewing CHM on a Network Drive..............557
Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x.................557
SVG Rendering Issues.............................................................................................................558
MSXML 4.0 Transformation Issues..........................................................................................558

Glossary.......................................................................................................................................559
Welcome to the User Manual of Oxygen XML Developer plugin 17.0!

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

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

This user guide is focused mainly at describing features, functionality and application interface to help you get started in no time.
# Key Features and Benefits of Oxygen XML Developer plugin  
**Eclipse plugin**

<table>
<thead>
<tr>
<th>Feature/Functionality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform availability: Windows, OS X, Linux, Solaris</td>
<td>Non blocking operations, you can perform validation and transformation operations in background</td>
</tr>
<tr>
<td>Closely integration of the DITA Open Toolkit for generating DITA output</td>
<td>Support for latest versions of document frameworks: DocBook and TEI.</td>
</tr>
<tr>
<td>Support for XML, XML Schema, Relax NG, Schematron, DTD, NVDL schemas, XSLT, XSL-FO, WSDL, XQuery, HTML, CSS</td>
<td>Support for XML, CSS, XSLT, XSL-FO.</td>
</tr>
<tr>
<td>Validate XML Schema schemas, Relax NG schemas, DTD’s, Schematron schemas, NVDL schemas, XQuery, HTML and CSS</td>
<td>Manual and automatic validation of XML documents against XML Schema schemas, Relax NG schemas, DTD’s, Schematron schemas, and NVDL schemas</td>
</tr>
<tr>
<td>Multiple built-in validation engines (Xerces, libxml, MSXML 4.0, MSXML.NET) and support for custom validation engines (Saxon SA, XSV, SQC).</td>
<td>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.</td>
</tr>
<tr>
<td>Visual schema editor with full and logical model views</td>
<td>Generate HTML documentation from XML Schemas</td>
</tr>
<tr>
<td>Ready to use FOP support to generate PDF or PS documents</td>
<td>XInclude support</td>
</tr>
<tr>
<td>Context sensitive content assistant driven by XML Schema, Relax NG, DTD, NVDL or by the edited document structure enhanced with schema annotation presenter</td>
<td>New XML document wizards to easily create documents specifying a schema or a DTD</td>
</tr>
<tr>
<td>XML Catalog support</td>
<td>Unicode support</td>
</tr>
<tr>
<td>Conversions from DTD, Relax NG schema or a set of documents to XML Schema, DTD or Relax NG schema</td>
<td>Syntax coloring for XML, DTD, Relax NG compact syntax, Java, C++, C, PHP, Perl, etc</td>
</tr>
<tr>
<td>Easy error tracking - locate the error source by clicking on it</td>
<td>Easy configuration for external FO Processors</td>
</tr>
<tr>
<td>Apply XSLT and FOP transformations</td>
<td>XPath search, evaluation and debugging support</td>
</tr>
<tr>
<td>Preview transformation results as XHTML or XML or in your browser</td>
<td>Support for document templates to easily create and share documents</td>
</tr>
<tr>
<td>Import data from a database, Excel, HTML or text file</td>
<td>Convert database structure to XML Schema</td>
</tr>
<tr>
<td>Batch validate selected files in project</td>
<td>Canonicalize and sign documents</td>
</tr>
<tr>
<td>Configurable actions key bindings</td>
<td>Associate extensions with editors provided by the Oxygen XML Developer plugin.</td>
</tr>
<tr>
<td>XSLT Debugger with Backmapping support</td>
<td>XSLT Profiler</td>
</tr>
<tr>
<td>XQuery Debugger with Backmapping support</td>
<td>XQuery Profiler</td>
</tr>
<tr>
<td>Model View</td>
<td>Attributes View</td>
</tr>
<tr>
<td>XQuery 1.0 support</td>
<td>WSDL analysis and SOAP requests support</td>
</tr>
<tr>
<td>XSLT 2.0 / 3.0 full support</td>
<td>XPath 2.0 / 3.0 execution and debugging support</td>
</tr>
<tr>
<td>Document folding</td>
<td>Spell checking supporting English, German and French including locals</td>
</tr>
<tr>
<td>XSLT refactoring actions</td>
<td>Generate large sets of sample XML instances from XML Schema</td>
</tr>
<tr>
<td>Pretty-printing of XML files</td>
<td>Drag&amp;drop support</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Outline view in sync with a non well-formed document</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2

Installation

Topics:

- Installation Options for Oxygen XML Developer plugin
- Install Oxygen XML Developer plugin on Windows
- Install Oxygen XML Developer plugin on Mac OS X
- Install Oxygen XML Developer plugin on Linux
- Site-wide Deployment
- Obtaining and Registering a License Key for Oxygen XML Developer plugin
- Setting Up a Floating License Server for Oxygen XML Developer plugin
- Transferring or Releasing a License Key
- Upgrading Oxygen XML Developer plugin
- Uninstalling Oxygen XML Developer plugin

The platform requirements and installation instructions are presented in this chapter.
Installation Options for Oxygen XML Developer plugin

Choosing an installer
You have a choice of installers;
• The Update Site installer
• The Zip archive installer

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

Choosing a license option
You must obtain and register a license key to run Oxygen XML Developer plugin.
You can choose from two kinds of license:
• A named-person license, which can be used by a single person on multiple computers.
• A floating license, which can be used by different people at different times. Only one person can use a floating license at a time.

Upgrading, transferring, and uninstalling.
You can also upgrade Oxygen XML Developer plugin, transfer a license, or uninstall Oxygen XML Developer plugin.

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

Install Oxygen XML Developer plugin on Windows

Choosing an installer
You can install Oxygen XML Developer plugin on Windows using one of the following methods:
• Install using the Update Site method.
• Install using the Zip archive method.

System Requirements
System requirements for a Windows install:
Operating systems
CPU
• Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 GHz
• Recommended - Dual Core class processor
Memory
• Minimum - 2 GB of RAM
• Recommended - 4 GB of RAM
Storage
• Minimum - 400 MB free disk space
Java
On Eclipse, Oxygen XML Developer plugin uses the same Java Virtual Machine as the copy of Eclipse it is running in.

Eclipse 3.6 - 4.5 Plugin Installation - The Update Site Method
Installation procedure for the Eclipse plugin in Eclipse 3.6 - 4.5 with the Update Site method.
1. Start Eclipse.
2. Go to Help > Install New Software... > Available Software.
3. Click Add ... in the Available Software dialog box.
5. Click OK.
6. Select the oXygen XML Developer checkbox.
7. Click Next > and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to File > New > Other... and choose oXygen XML Developer > XML Project.
10. When prompted for a license key, enter the license information received in the registration email.

Note that if you already have a native version of Oxygen XML Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Eclipse 3.6 - 4.5 Plugin Installation - The Zip Archive Method
The steps for installing the Eclipse plugin in Eclipse 3.6 - 4.5 with the Zip Archive method.
1. Download the zip archive with the Eclipse plugin.
2. Unzip the downloaded zip archive in the dropins subdirectory of the Eclipse install directory.
3. Restart Eclipse.
4. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to File > New > Other... and choose oXygen XML Developer > XML Project.
5. When prompted for a license key, enter the license information received in the registration email.

Note that if you already have a native version of Oxygen XML Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Install Oxygen XML Developer plugin on Mac OS X

Choosing an installer
You can install Oxygen XML Developer plugin on Mac OS X using one of the following methods:

- Install using the Update Site method.
- Install using the Zip archive method.

System Requirements
System requirements for a Mac OS X install:

Operating system
Mac OS X version 10.5 64-bit or later
CPU
- Minimum - Intel-based Mac, 1 GHz
- Recommended - Dual Core class processor

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

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

Java
On Eclipse, Oxygen XML Developer plugin uses the same Java Virtual Machine as the copy of Eclipse it is running in.

Eclipse 3.6 - 4.5 Plugin Installation - The Update Site Method
Installation procedure for the Eclipse plugin in Eclipse 3.6 - 4.5 with the Update Site method.

1. Start Eclipse.
2. Go to Help > Install New Software... > Available Software.
3. Click Add ... in the Available Software dialog box.
5. Click OK.
6. Select the oXygen XML Developer checkbox.
7. Click Next > and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to File > New > Other... and choose oXygen XML Developer > XML Project.
10. When prompted for a license key, enter the license information received in the registration email.

Note that if you already have a native version of Oxygen XML Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Eclipse 3.6 - 4.5 Plugin Installation - The Zip Archive Method
The steps for installing the Eclipse plugin in Eclipse 3.6 - 4.5 with the Zip Archive method.

1. Download the zip archive with the Eclipse plugin.
2. Unzip the downloaded zip archive in the dropins subdirectory of the Eclipse install directory.
3. Restart Eclipse.
4. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to File > New > Other... and choose oXygen XML Developer > XML Project.
5. When prompted for a license key, enter the license information received in the registration email.

Note that if you already have a native version of Oxygen XML Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.
Install Oxygen XML Developer plugin on Linux

Choosing an installer
You can install Oxygen XML Developer plugin on Linux using any of the following methods:

• Install using the Update Site method.
• Install using the Zip archive method.

System Requirements
System requirements for a Linux install:

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

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

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

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

Java
On Eclipse, Oxygen XML Developer plugin uses the same Java Virtual Machine as the copy of Eclipse it is running in.

Eclipse 3.6 - 4.5 Plugin Installation - The Update Site Method
Installation procedure for the Eclipse plugin in Eclipse 3.6 - 4.5 with the Update Site method.

1. Start Eclipse.
2. Go to Help > Install New Software... > Available Software.
3. Click Add ... in the Available Software dialog box.
5. Click OK.
6. Select the oXygen XML Developer checkbox.
7. Click Next > and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to File > New > Other... and choose oXygen XML Developer > XML Project.
10. When prompted for a license key, enter the license information received in the registration email.

   Note that if you already have a native version of Oxygen XML Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Eclipse 3.6 - 4.5 Plugin Installation - The Zip Archive Method
The steps for installing the Eclipse plugin in Eclipse 3.6 - 4.5 with the Zip Archive method.

1. Download the zip archive with the Eclipse plugin.
2. Unzip the downloaded zip archive in the dropins subdirectory of the Eclipse install directory.

3. Restart Eclipse.

4. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to File > New > Other... and choose oXygen XML Developer > XML Project.

5. When prompted for a license key, enter the license information received in the registration email.
   
   Note that if you already have a native version of Oxygen XML Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

---

**Site-wide Deployment**

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

**Creating custom default options**

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

**Creating default project files**

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

**Shared project files**

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

**Using floating licenses**

If you have a number of people using Oxygen XML Developer plugin on a part-time basis or in different time zones, you can use a floating license so that multiple people can share a license.

---

**Obtaining and Registering a License Key for Oxygen XML Developer plugin**

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

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

**Choosing a license type**

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

1. A named-user license may be used by a single named user on one or more computers. Named-user licenses are not transferable to a new named user. If you order multiple named-user licenses, you will receive a single license key good for a specified number of named users. It is your responsibility to keep track of the named users that each license is assigned to.

2. A floating license may be used by any user on any machine. However, the total number of copies of Oxygen XML Developer plugin in use at one time must not be more than the number of floating licenses available. A user who runs two different distributions of Oxygen XML Developer plugin (for example Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single floating license.

Obtaining a license

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

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

Register a named-user license

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

1. Save a backup copy of the message containing the new license key.
   If this is a new install of Oxygen XML Developer plugin, the registration dialog box is displayed. If the registration dialog box is not displayed, go to Window (Eclipse on Mac OSX) and choose Preferences > oXygen XML Developer and click on the Register... button.

   ![License Registration Dialog Box](image)

   Figure 1: License Registration Dialog Box

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

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

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

To register licenses using a text file:

1. Copy the license key to a file named licensekey.txt and place it in the lib sub-folder of the Oxygen XML Developer plugin installation folder.

To register licenses using an XML file:

1. Register the license on one computer using the normal license registration procedure.
2. Copy the license.xml file from the Oxygen XML Developer plugin preferences directory on that computer to lib sub-folder of the installation folder on each installation to be registered.

Registering a floating license

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

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

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

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

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

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

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

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

If a floating license is available, it is registered in Oxygen XML Developer plugin. To display the license details, open the Preferences dialog box. If a floating license is not available, you will get a message listing the users currently using floating licenses.

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

1. Contact your server administrator to get network address and login details for the license server.
2. Start the Eclipse platform.
3. *Open the Preferences dialog box* and click the **Register** button. The license registration dialog box is displayed.
4. Choose **Use a license server** as licensing method.
5. Select **HTTP/HTTPS Server** as server type.
6. In the **URL** field enter the address of the license server. The URL address has the following format:
   
   `http://hostName:port;oXygenLicenseServlet/license-servlet`

7. Complete the **User** and **Password** fields.
8. Click the **OK** button.

If a floating license is available, it is registered in Oxygen XML Developer plugin. To display the license details, *open the Preferences dialog box*. If a floating license is not available, you will get a message listing the users currently using floating licenses.

**Release a Floating License**

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

- The connection with the license server is lost.
- You exit the application running on your machine, and no other copies of Oxygen XML Developer plugin running on your machine are using your floating license.
- You register a named user license with your copy of Oxygen XML Developer plugin, and no other copies of Oxygen XML Developer plugin running on your machine are using your floating license.

**Setting Up a Floating License Server for Oxygen XML Developer plugin**

**Determine if you need to set up a license server**

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

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

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

**Activating Floating License Keys**

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

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

**Split or combine license keys to work with your license servers**

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

1. Make sure that Apache Tomcat 5.5 or higher is running on the machine you have selected to be the license server. To get it, go to http://tomcat.apache.org.

2. Download the Web ARchive (.war) license servlet from the Oxygen XML Developer plugin website.

3. Configure Tomcat to use a security Realm element. Please refer to the Tomcat Documentation for more information.

4. Edit the tomcat-users.xml file from your Tomcat installation and configure one user for each of the following roles: standard, admin, and manager.

5. Go to the Tomcat Web Application Manager page and log-in with the user you configured with the manager role. In the WAR file to deploy section, choose the WAR file and click the Deploy button. The oXygen License Servlet is now up and running, but the license keys are not yet registered.

6. Activate the license key. This process involves binding your license key to your license server deployment. Once the process is completed you cannot activate the license with another license server. Follow these steps to activate the license:
   
   a. Access the license servlet by following the link provided by the Tomcat Web Application Manager page. If prompted for authentication, use the credentials configured for the admin or manager users.

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

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

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

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

   c. Press **Activate**.

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

7. By default, the license server logs its activity in the /usr/local/tomcat/logs/oxygenLicenseServlet.log file. To change the log file location, edit the log4j.appender.R2.File property from the...
Manual Activation Procedure

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

Report Page

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

It displays the following real time information:

- **License load** - A graphical indicator that shows how many licenses are available. When the indicator turns red, there are no more licenses available.

- **Floating license server status** - General information about the license server status, including the following information:
  - server start time
  - license count
  - rejected and acknowledged requests
  - average usage time
  - license refresh and timeout intervals
  - location of the license key
  - server version

- **License key information** - License key data, including the following information:
  - licensed product
  - registration name
  - company name
  - license category
  - number of floating users
  - Maintenance Pack validity

- **Current license usage** - Lists all currently acknowledged users, including the following information:
  - user name
  - date and time when the license was granted
  - name and IP address of the computer where Oxygen XML Developer plugin runs
  - MAC address of the computer where Oxygen XML Developer plugin runs

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

Replacing a Floating License Key

The following procedure assumes that your Oxygen XML Developer plugin floating license servlet contains a previously activated license key. The following procedure contains instructions for replacing the activated license key with another one. The goal of the procedure is to minimize the license servlet down-time during the activation step of the new license key.
This is useful if, for instance, you want to upgrade your existing license to the latest version or if you receive a new license key that accommodates a different number of users.

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

1. Access the license servlet by following the link provided by the Tomcat Web Application Manager page.
2. Click the Replace license key link. This will take you to a page that contains details about the license currently in use.
3. Click the Yes button to begin the replacement procedure.
   
   Note: During the replacement procedure, new instances of Oxygen XML Developer plugin cannot be licensed by the servlet.

4. Paste the new floating license key in the displayed form, then click Submit. The browser used in the process needs to have Internet access.
   
   You will be redirected to an online form hosted on the Oxygen XML Developer plugin website. This form is pre-filled with an activation code that uniquely identifies your license server deployment and your license key.
   
   Note: If, for some reason, your browser does not take you to this activation form, refer to the Manual Activation Procedure.

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

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

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

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

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

1. Download the license server installation kit for Windows from the Oxygen XML Developer plugin website.
2. Run the downloaded installer and follow the on-screen instructions.

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

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

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

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

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

3. Manually add the oXygenLicenseServer.exe file in the Windows Firewall list of exceptions. Go to Control Panel > System and Security > Windows Firewall > Allow a program or feature through Windows Firewall > Allow another program and browse for oXygenLicenseServer.exe from the oXygen License Server installation folder.

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

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

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

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

Make sure that you have installed a 32-bit Java SE from Oracle (or Sun) on the system: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

When Started, the Windows Service Reports "Error 1067: The Process Terminated Unexpectedly"
This error message appears if the Windows service launcher quits immediately after being started.

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

Setting up a Floating License Server Running as a Standalone Process Using a Platform-independent Distribution
This installation method can be used for running the license server on any platform where a Java virtual machine can run (OS X, Linux/Unix, Windows).
Steps for Installing the Floating License Server as a Standalone Process with a Zip Archive

1. Ensure that a Java runtime version 6 or later is installed on the server machine.
2. Download the license server installation kit for your platform from the Oxygen XML Developer plugin website.
3. Unzip the installation kit into a new folder.
4. Start the server using the startup script from a command line console.
   
   The startup script is called `licenseServer.sh` for OS X and Unix/Linux or `licenseServer.bat` for Windows. The following parameters are accepted:
   
   - `licenseDir` - The path of the directory where the license files will be placed. The default value is `license`.
   - `port` - The TCP port number used to communicate with Oxygen XML Developer plugin instances. The default value is `12346`.

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

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

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

Transferring or Releasing a License Key

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

1. Open the Preferences dialog box and click Register.
   
   The license registration dialog box is displayed.

2. The license key field should be empty (this is normal). If it is not empty, delete any text in the field.

3. Make sure the option Use a license key is selected.

4. Click OK.
   
   A dialog box is displayed asking if you want to reset your license key.

5. Select between falling back to the license key entered previously (for the case of releasing a floating license and reverting to Named User license) and removing your license key from your user account on the computer using the Reset button.

   The Reset button erases all the licensing information. To complete the reset operation, close and restart Oxygen XML Developer plugin.
Upgrading Oxygen XML Developer plugin

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

Checking for New Versions of Oxygen XML Developer plugin

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

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

What is preserved during an upgrade

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

If you install over a previously installed version:

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

If you install in a new directory:

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

Upgrading the Eclipse Plugin

1. Uninstall the current version of Oxygen XML Developer plugin.
2. Download and install the new version using the appropriate instructions for your platform and the installation method you chose.
3. Restart the Eclipse platform.
4. Start the Oxygen XML Developer plugin to ensure that the application can start and that your license is recognized by the upgrade installation.
5. If you are upgrading from a minor version to a major version (for example, from 16.1 to 17.0) and you did not purchase a Maintenance Pack that covers the new major version (17.0), you will need to enter a new license for the new version (17) into the registration dialog box that is shown when the plugin is started.
Uninstalling Oxygen XML Developer plugin

Uninstalling the Eclipse plugin

⚠️ Caution:

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

1. Choose the menu option Help > About > Installation Details.
2. Select Oxygen XML Developer plugin from the list of plugins.
3. Choose Uninstall.
4. Accept the Eclipse restart.
5. If you also want to remove the user preferences you must remove the folder
   %APPDATA%\com.oxygenxml.developer on Windows (usually %APPDATA% has the value [user-home-dir]\Application Data)/the subfolder .com.oxygenxml.developer of the user home directory on Linux/the subfolder Library/Preferences/com.oxygenxml.developer of the user home folder on Mac OS X.
Chapter 3

Perspectives

Topics:
- Perspectives

This chapter describes the editing perspectives of Oxygen XML Developer plugin.
Perspectives

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

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

**Editor perspective**

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

**XSLT Debugger perspective**

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

**XQuery Debugger perspective**

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

**Database perspective**

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

**Oxygen XML Developer plugin XML Perspective**

To edit the content of your XML documents, use the `<oXygen/> XML` perspective (Window > Open Perspective > `<oXygen/> XML`).

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

**The Oxygen XML Developer plugin Custom Menu**

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

**The Oxygen XML Developer plugin Toolbar Buttons**

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

**The Editor Pane**

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

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

**The Outline View**

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

The outline view has the following functions: XML document overview, outline filters, modification follow-up, document structure change, document tag selection.
The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcards (*, ?) and separate multiple patterns with commas.

Figure 2: The Outline View

The Oxygen XML Developer plugin Text View

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

Figure 3: The Text View

The Oxygen XML Developer plugin Browser View

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

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

- Validate action
- Transform action
- Check Spelling in Files action
- Search References action
- SQL results

The view provides a toolbar with the following actions:

Remove actions

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

The actions available on the contextual menu are:

- ✗ Remove selected
  - Removes selected messages from the view.

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

Save Results ...
Saves the complete list of messages in a file in text format. For each message the included details are the same as the ones for the Copy action.

Save Results as XML
Saves the complete list of messages in a file in XML format. For each message the included details are the same as the ones for the Copy action.

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

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

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

Figure 6: The XPath Results View

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

• • - The XML documents icon
• • - The XSL stylesheets icon
• • - The XML Schema icon
• • - The Document Type Definition schemas icon
• • - The RELAX NG full syntax schemas icon
• • - The RELAX NG compact syntax schemas icon
• • - The Namespace-based Validation Dispatching Language schemas icon
• • - The XSL:FO documents icon
• • - The XQuery documents icon
• • - The WSDL documents icon
• • - The Schematron documents icon
• • - The JavaScript documents icon
XSLT Debugger Perspective

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

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

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

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

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

XQuery Debugger Perspective

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

The workspace is organized as follows:

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

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

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

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

Oxygen XML Developer plugin Database Perspective

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

This perspective offers database specific support for:

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

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

The non-XML capabilities are:

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

The XML capabilities are:

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

For a detailed feature matrix that compares the Academic, Professional and Enterprise editions of Oxygen XML Developer plugin please go to the Oxygen XML Developer plugin website.

Note: Only connections configured on relational data sources can be used to import data to XML or to generate XML schemas.
The perspective provides the following functional areas:

- **Main menu** - provides access to all the features and functions available within Oxygen XML Developer plugin.
- **Main toolbar** - provides easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function.
- **Editor area** - the place where you spend most of your time, reading, editing, applying markup and validating your documents.
- **Data Source Explorer** - provides browsing support for the configured connections.
- **Table explorer** - provides table content editing support for inserting new rows, deleting table rows, cell value editing, export to XML file.
Chapter 4

Editing Modes

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

- **Text** - this mode presents the source of an XML document.
- **Grid** - this mode displays an XML document as a structured grid of nested tables.
- **Design** - this mode is found in the schema editor and represents the schema as a diagram.
Text Editing Mode

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

Finding and Replacing Text in the Current File

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

You can use a number of advanced views depending on what you need to find in the document you are editing or in your entire project. The **Find All Elements/Attributes dialog box** allows you to search through the structure of the current document for elements and attributes.

The Find All Elements Dialog Box

To open the **Find All Elements** dialog box, go to **Edit > Find All Elements...**. It assists you in defining XML element / attribute search operations in the current document.

![Find All Elements Dialog Box](image)

---

**Figure 7: Find All Elements Dialog Box**

The dialog box can perform the following actions:

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

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

The following fields are available in the dialog box:

- **Element name** - the qualified name of the target element to search for. You can use the drop-down list to find an element or enter it manually. The drop-down list is populated with valid element names collected from the associated schema. To specify *any* element name, leave the field empty.
  
  **Note:** Use the qualified name of the element (<namespace prefix>:<element name>) when the document uses this element notation.

- **Element text** - the target element text to search for. The drop-down list beside this field allows you to specify that you are looking for an exact or partial match of the element text. For *any* element text, select **contains** in the drop-down list and leave the field empty. If you leave the field empty but select **equals** in the drop-down list, only elements with no text will be found. Select **not contains** to find all elements that do not include the specified text.
• **Attribute name** - the name of the attribute that must be present in the element. You can use the drop-down list to select an attribute or enter it manually. The drop-down list is populated with valid attribute names collected from the associated schema. For *any* or no attribute name, leave the field empty.

  **Note:** Use the qualified name of the attribute (<namespace prefix>:<attribute name>) when the document uses this attribute notation.

• **Attribute value** - the drop-down list beside this field allows you to specify that you are looking for an exact or partial match of the attribute value. For *any* or no attribute value, select *contains* in the drop-down list and leave the field empty. If you leave the field empty but select *equals* in the drop-down list, only elements that have at least an attribute with an empty value will be found. Select *not contains* to find all elements that have attributes without a specified value.

• **Case sensitive** - when this option is checked, operations are case-sensitive

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

**Regular Expressions Syntax**

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

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

Other notable difference:

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

**Grid Editing Mode**

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

In case you are a non-technical user, you are able to modify the text content of the edited document without working with the XML tags directly. You can expand and collapse the tables using the mouse cursor and also display or hide the elements of the document as nested. The document structure can also be changed easily with drag and drop operations on the grid components. To zoom in and out, use **Ctrl + (Command + on OS X)**, **Ctrl - (Command - on OS X)**, **Ctrl 0 (Command 0 on OS X)** or **Ctrl Scroll Forward (Command Scroll Forward on OS X)** / **Ctrl Scroll Backwards (Command Scroll Backwards on OS X)**.
Figure 8: The Grid Editor

To switch back from the Grid mode to the Text or Author mode, use the Text and Grid buttons from the bottom of the editor.

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

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

Figure 9: Content Completion in Grid Editor

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

Layouts: Grid and Tree

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

Figure 10: Grid Layout

The other layout mode is tree-like. It does not create any tables and it only presents the structure of the document.
To switch between the two modes, go to the contextual menu > Grid mode/Tree mode.

**Grid Move Navigation**

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

**Table 1: Shortcuts in the Grid Mode**

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>Moves the caret to the next editable value in a table row.</td>
</tr>
<tr>
<td>Shift Tab</td>
<td>Moves the caret to the previous editable value in a table row.</td>
</tr>
<tr>
<td>Enter</td>
<td>Begins editing and lets you insert a new value. Also commits the changes after you finish editing.</td>
</tr>
<tr>
<td>Up Arrow/Page Up</td>
<td>Navigates toward the beginning of the document.</td>
</tr>
<tr>
<td>Down Arrow/Page Down</td>
<td>Navigates toward the end of the document.</td>
</tr>
<tr>
<td>Shift</td>
<td>Used in conjunction with the navigation keys to create a continuous selection area.</td>
</tr>
<tr>
<td>Ctrl (Command on OS X) key</td>
<td>Used in conjunction with the mouse cursor to create discontinuous selection areas.</td>
</tr>
</tbody>
</table>

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

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

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

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

- **Expand All**
  Expands the selection and all its children.
Collapse All
Collapses the selection and all its children.

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

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

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

Specific Grid Actions

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

Sorting a Table Column

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

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

Inserting a Row in a Table

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

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

Inserting a Column in a Table

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

Clearing the Content of a Column

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

Adding Nodes

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

- Insert before
- Insert after
- Append child

Duplicating Nodes

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

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

Start and Stop Editing a Cell Value

To edit the value of a cell, simply select the grid cell and press (Enter).

To stop editing a cell value, press (Enter) again.

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

Drag and Drop in the Grid Editor

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

You can do the following with drag and drop:

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

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

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

Copy and Paste in the Grid Editor

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

You can select discontinuous regions of nodes and place them in the clipboard using the copy action. To deselect one of the selected fragments, use Ctrl Click (Command Click on OS X). Pasting these nodes relative to the current selected cell may be done in two ways: just below (after) as a brother, which is the default behavior, or as the last child of the selected cell.

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

The nodes copied from the Grid editor can also be pasted into the Text editor or other applications. When copying from the Grid into the Text editor or other text based applications, the inserted string represents the nodes serialization. The nodes from tables can be copied using HTML or RTF in table format. The resulting cells contain only the concatenated values of the text nodes.
In the Grid editor you can paste well-formed XML content or tab separated values from other editors. If you paste XML content, the result will be the insertion of the nodes obtained by parsing this content.

If the pasted text contains multiple lines of tab separated values it can be considered as a matrix of values. By pasting this matrix of values into the Grid editor the result will be a matrix of cells. If the operation is performed inside existing cells, the existing values will be overwritten and new cells will be created when needed. This is useful, for example, when trying to transfer data from Excel like editors into the Grid editor.
Bidirectional Text Support in Grid Mode

If you are editing documents employing a different text orientation, you can change the way the text is rendered and edited in the grid cells by using the Ctrl Shift O (Command Shift O on OS X) shortcut to switch from the default left to right text orientation to the right to left orientation.

**Note:** This change applies only to the text from the cells, and not to the layout of the grid editor.

---

**Figure 15: Default left to right text orientation**

```
<?xml version="1.0" encoding="UTF-8"?>
sample
text
1. Cuando el món vol conversar, parla Unicode
2. Quan el món vol conversar, parla Unicode
3. Ha a világ beszélni akar, azt Unicode-ül mondja
4. Quando il mondo vuole comunicare, parla Unicode
5. 世界的に話すなら、Unicodeです。
6. 세계를 통한 대화, 유니코드로 하십시오
7. Når verden vil snakke, snakker den Unicode
8. Når verden vil snakke, snakker den Unicode
9. Når verden vil snakke, talar ho Unicode
```

**Figure 16: Right to left text orientation**

---
This chapter explains the editor types available in Oxygen XML Developer plugin and how to work with them for editing different types of documents.
Working with Unicode

Unicode provides a unique number for every character, independent of the platform and language. Unicode is an internationally recognized standard, adopted by industry leaders. The Unicode is required by modern standards such as XML, Java, ECMA-Script (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646.

It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends. Incorporating Unicode into client-server or multi-tiered applications and websites offers significant cost savings over the use of legacy character sets.

As a modern XML Editor, Oxygen XML Developer plugin provides support for the Unicode standard enabling your XML application to be targeted across multiple platforms, languages, and countries without re-engineering. Internally, the Oxygen XML Developer plugin XML Editor uses 16-bit characters covering the Unicode Character set.

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

Tip: On windows, you can enable the support for CJK (Chinese, Japanese, Korean) languages from Control Panel / Regional and Language Options / Languages / Install files for East Asian languages.

Opening and Saving Unicode Documents

When loading documents, Oxygen XML Developer plugin receives the encoding of the document from the Eclipse platform. This encoding is then used to instruct the Java Encoder to load support for and to save the document using the specified code chart.

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

To edit documents written in Japanese or Chinese, change the font to one that supports the specific characters (a Unicode font). For the Windows platform, Arial Unicode MS or MS Gothic is recommended. Do not expect WordPad or Notepad to handle these encodings. Use Internet Explorer or Word to examine XML documents.

When a document with a UTF-16 encoding is edited and saved in Oxygen XML Developer plugin, the saved document has a byte order mark (BOM) which specifies the byte order of the document content. The default byte order is platform-dependent. That means that a UTF-16 document created on a Windows platform (where the default byte order mark is UnicodeLittle) has a different BOM than a UTF-16 document created on a Mac OS platform (where the byte order mark is UnicodeBig). The byte order and the BOM of an existing document are preserved when the document is edited and saved.

Creating, Opening, and Closing Documents

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

Creating Documents

This section details the procedures available for creating new documents.

Oxygen XML Developer plugin New Document Wizard

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

The Oxygen XML Developer plugin plugin installs a series of Eclipse wizards for easy creation of documents. If you use these wizards, Oxygen XML Developer plugin automatically completes the following details:
• The system ID, or schema location of a new XML document.
• The minimal markup of a DocBook article, or the namespace declarations of a Relax NG schema.

1. To create a document, either select **File > New > Other > Ctrl N (Command N on OS X) > oXygen**, or click the **New** button on the toolbar.
   
   The **New** wizard is displayed.

2. Select a document type.

3. Click the **Next** button.

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

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

4. Type a name for the new document and press the **Next** button.

5. If you select **Customize**, Oxygen XML Developer plugin opens the following dialog box. You can customize different options depending on the document type you select.

![New XML Document Dialog Box](image)

**Figure 17: New XML Document Dialog Box**

• **Schema URL** - Specifies the path to the schema file. When you select a file, Oxygen XML Developer plugin analyzes its content and tries to fill the rest of the dialog box.

• **Schema type** - Allows you to select the schema type. The following options are available: XML Schema, DTD, RelaxNG XML syntax, RelaxNG compact syntax, and NVDL.

• **Public ID** - Specifies the PUBLIC identifier declared in the document prolog.

• **Namespace** - Specifies the document namespace.

• **Prefix** - Specifies the prefix for the namespace of the document root.

• **Root Element** - Populated with elements defined in the specified schema, enables selection of the element used as document root.
• **Description** - Shows a small description of the selected document root.
• **Add optional content** - If you select this option, the elements, and attributes defined in the XML Schema as optional, are generated in the skeleton XML document.
• **Add first Choice particle** - If you select this option, Oxygen XML Developer plugin generates the first element of an `xs:choice` schema element in the skeleton XML document. Oxygen XML Developer plugin creates this document in a new editor panel when you click **OK**.

![New XSL Document Dialog Box](image)

**Figure 18: New XSL Document Dialog Box**

• **Stylesheet version** - Allows you to select the Stylesheet version number. You can select from: 1.0, 2.0, and 3.0.
• **Add documentation annotations** - Adds annotation for XSL components.
• **Open file for editing when done** - when you press **Finish**, Oxygen XML Developer plugin opens the newly created file.
Figure 19: New XML Schema Document Dialog Box

- **Default XML Schema version** - Uses the XML Schema version defined in the XML Schema preferences page.
- **XML Schema 1.0** - Sets the minVersion attribute to 1.0 and the maxVersion attribute to 1.1.
- **XML Schema 1.1** - Sets the minVersion attribute to 1.1.
- **Target namespace** - specifies the schema target namespace.
- **Namespace prefix declaration table** - contains namespace prefix declarations. To manage table information, use the New and Delete buttons.

Tip: For further details on how you can set the version of an XML Schema, go to Setting the XML Schema Version.
Creating Documents Based on Templates

The New wizard enables you to select predefined templates or custom templates. Custom templates are created in previous sessions or by other users.

The list of templates presented in the dialog includes:

- Document Types templates - Templates supplied with the defined document types.
- User defined templates - You can add template files to the templates folder of the Oxygen XML Developer plugin install directory. You can also specify another directory to use for templates. Open the Preferences dialog box and go to Editor > Templates > Document Templates to specify a custom templates folder.

1. Go to menu File > New > Other > oXygen > New From Templates.
2. Select a document type.
3. Type a name for the new document and press the Next button.
4. Press the Finish button.

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

Document Templates

Templates are documents that have a predefined structure. They provide the starting point from which you can build new documents rapidly, based on the same characteristics (file type, prolog, root element, existing content). Oxygen XML Developer plugin offers a rich set of templates for a number of XML applications. You may also create your own templates and share them with others.

To configure or add templates, open the Preferences dialog box and go to Editor > Templates > Document Templates.

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

You can save the document you are editing with one of the following actions:

- File > Save.
- File > Save As - displays the Save As dialog, used either to name and save an open document to a file or to save an existing file with a new name.
- File > Save All - saves all open documents.

Opening and Saving Remote Documents via FTP/SFTP

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

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

To avoid conflicts with other users when you edit a resource stored on a SharePoint server, you can Check Out the resource.

To improve the transfer speed, the content exchanged between Oxygen XML Developer plugin and the HTTP/WebDAV server is compressed using the GZIP algorithm.

The current WebDAV Connection details can be saved using the Database Perspective button and then used in the Data Source Explorer view.

The Open Using FTP/SFTP/WebDAV Dialog Box

To access the Open using FTP/SFTP/WebDAV dialog box, go to File > Open URL ... menu, then choose the Browse for remote file option from the drop-down action list.
The displayed dialog box is composed of several parts:

- The editable combo box, in which it can be specified directly the URL to be opened or saved.
  
  **Tip:** You can type in here an URL like ftp://anonymous@some.site/home/test.xml if the file is accessible through anonymous FTP.

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

- The *Identification* section contains the access credentials. If you want to browse for a file on a server, you have to specify the user and password. This information is bound to the selected URL displayed in the *File URL* combo box, and used further in opening/saving the file. If the check box *Save* is selected, then the user and password are saved between editing sessions. The password is kept encrypted into the options file.

  **Note:** Your password is well protected. In the case the options file is used on other machine by a user with a different user name the password will become unreadable, since the encryption is user-name dependent. This is also true if you add URLs having user and password to your project.

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

  **Tip:** When accessing a FTP server, you need to specify only the protocol and the host, like: ftp://server.com, or if using a nonstandard port: ftp://server.com:7800/.
By pressing the **Browse** button, the directory listing will be shown in the component below. When **Autoconnect** is selected, every time the dialog box is displayed, the browse action will be performed.

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

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

### Changing File Permissions on a Remote FTP Server

Some FTP servers allow the modification of permissions of the files served over the FTP protocol. This protocol feature is accessible directly in the FTP file browser dialog box by right-clicking on a tree node and selecting the **Change permissions** menu item.

In this dialog box, the usual Unix file permissions **Read**, **Write**, and **Execute** are granted or denied for the file owner, owner group, and the rest of the users. The aggregate number of permissions is updated in the **Permissions** text field when it is modified with one of the check boxes.

### WebDAV over HTTPS

If you want to access a WebDAV repository across an insecure network, Oxygen XML Developer plugin allows you to load and save the documents over the HTTPS protocol (if the server understands this protocol) so that any data exchange with the WebDAV server is encrypted.

When a WebDAV repository is first accessed over HTTPS, the server hosting the repository will present a security certificate as part of the HTTPS protocol, without any user intervention. Oxygen XML Developer plugin will use this certificate to decrypt any data stream received from the server. For the authentication to succeed you should make sure the security certificate of the server hosting the repository can be read by Oxygen XML Developer plugin. This means that Oxygen XML Developer plugin can find the certificate in the key store of the Java Runtime Environment in which it runs. You know the server certificate is not in the JRE key store if you get the error **No trusted certificate found** when trying to access the WebDAV repository.

### Troubleshooting HTTPS

When Oxygen XML Developer plugin cannot connect to an HTTPS-capable server, most likely there is no certificate set in the **Java Runtime Environment (JRE)** that Oxygen XML Developer plugin runs into. The following procedure describes how to:

- export a certificate to a local file using any HTTPS-capable Web browser (for example Internet Explorer)
- import the certificate file into the JRE using the keytool tool that comes bundled with Oxygen XML Developer plugin

1. Export the certificate into a local file
   a) Point your HTTPS-aware Web browser to the repository URL.

      If this is your first visit to the repository it will be displayed a security alert stating that the security certificate presented by the server is not trusted.
b) Go to menu **Tools > Internet Options**.
   
   **Internet Options** dialog box is opened.
   
   c) Select **Security** tab.
   
   d) Select **Trusted sites** icon.
   
   e) Press **Sites** button.
   
   This will open **Trusted sites** dialog box.
   
   f) Add repository URL to **Websites** list.
   
   g) Close the **Trusted sites** and **Internet Options** dialog boxes.
   
   h) Try again to connect to the same repository URL in Internet Explorer.
   
   The same error page as above will be displayed.
   
   i) Select **Continue to this website** option.
   
   A clickable area with a red icon and text **Certificate Error** is added to Internet Explorer address bar.
   
   j) Click on **Certificate Error** area.
   
   A dialog box containing a **View certificates** link is displayed.
   
   k) Click on **View certificates** link.
   
   **Certificate** dialog box is displayed.
   
   l) Select **Details** tab of **Certificate** dialog box.
   
   m) Press **Copy to File** button.
   
   **Certificate Export Wizard** is started.
   
   n) Follow indications of wizard for DER encoded binary X.509 certificate. Save certificate to local file **server.cer**.

2. Import the local file into the JRE running Oxygen XML Developer plugin.
   
   a) Open a text-mode console with administrative rights.
   
   b) Go to the **lib/security** directory of the JRE running Oxygen XML Developer plugin. You find the home directory of the JRE in the **java.home** property that is displayed in the **About** dialog box (**Installation Details > Configuration**). On Mac OS X systems, the **lib/security** directory is usually located in

   /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home directory.
   
   c) Run the following command:

   ```
   ..\..\bin\keytool -import -trustcacerts -file server.cer -keystore cacerts
   ```
The `server.cer` file contains the server certificate, created during the previous step. `keytool` requires a password before adding the certificate to the JRE `keystore`. The default password is `changeit`. If somebody changed the default password then he is the only one who can perform the import.

Note: To make Oxygen XML Developer plugin accept a certificate even if it is invalid, open the Preferences dialog box, go to Connection settings > HTTP(S)/WebDAV, and enable the Automatically accept a security certificate, even if invalid option.

Tip: If you need to import multiple certificates, you need to specify a different alias for each additional imported certificate with the `–alias` command line argument, like in the following example:

```
..\..\bin\keytool -import -alias myalias1 -trustcacerts -file server1.cer -keystore cacerts
..\..\bin\keytool -import -alias myalias2 -trustcacerts -file server2.cer -keystore cacerts
```


**Opening the Current Document in System Application**

To open the currently edited document in the associated system application, use the View in Browser/System Application action that is available in the XML menu. If you want to open XML files in a specific internet browser, instead of the associated system application, you can specify the internet browser to be used. To do so, open the Preferences dialog box, then go to General > Web Browser. This will take precedence over the default system application settings.

**Closing Documents**

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

- Go to menu **File > Close (Ctrl F4 (Command F4 on OS X))**: Closes only the selected tab. All other tab instances remain opened.
- Go to menu **File > Close All (Ctrl Shift F4 (Command Shift F4 on OS X))**: If you try to close a modified or a newly created document, you are first prompted to save it.
- Click **Close** in the contextual menu of an open tab to close it.
- Click **Close Other Files** in the contextual menu of an open tab to close all the open tabs except the selected one.
- Click **Close All** in the contextual menu of an open tab to close all open tabs.

**The Contextual Menu of the Editor Tab**

The contextual menu is available when clicking the current editor tab label. It shows the following actions:

- **Close**
  - Closes the current editor.

- **Close Other Files**
  - Closes all opened editor but the one you are currently viewing.

- **Close All**
  - Closes all opened editors.

- **Reopen last closed editor**
  - Reopens the last closed editor.

- **Maximize/Restore Editor Area**
  - Collapses all the side views and spans the editing area to cover the entire width of the main window.

- **Add to project**
  - Adds the file you are editing to the current project.

- **Add all to project**
  - Adds all the opened files to the current project.
**Copy Location**

Copies the disk location of the file.

**Show in Explorer (Show in Finder on OS X)**

Opens the Explorer to the file path of the file.

---

**Viewing File Properties**

In the **Properties** view, you can quickly access information about the current edited document like:

- character encoding
- full path on the file system
- schema used for content completion and document validation
- document type name and path
- associated transformation scenario
- file's read-only state
- bidirectional text (left to right and right to left) state
- total number of characters in the document
- line width
- indent with tabs state
- indent size

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

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

---

**Grouping Documents in XML Projects**

This section explains how to create and work with projects.

**Using the Navigator View**

The **Navigator** 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 23: The Navigator View](image)
The following actions are grouped in the upper right corner:

- **Collapse All**
  Collapses all project tree folders. You can also collapse/expand a project tree folder if you select it and press the **Enter** key or **Left Arrow** to collapse and **Right Arrow** to expand.

- **Link with Editor**
  When selected, the project tree highlights the currently edited file, if it is found in the project files.

  **Note:** This button is disabled automatically when you move to the **Debugger** perspective.

**View Menu**
Drop down list that contains various settings.

The files are usually organized in an XML project as a collection of folders. There are two types of resources displayed in the **Navigator** view:

- **Physical folders and files** - marked with the operating system-specific icon for folders (usually a yellow icon on Windows and a blue icon on Mac OS X). These folders and files are mirrors of real folders or files that exist in the local file system. They are created or added to the project by using contextual menu actions (such as **New > File** and **New > Folder**). Also, the contextual menu action **Delete** can be used to remove them from the project and local file system.

- **Shortcut folders and files** - the icons for file and folder shortcuts are displayed with a shortcut symbol. They are created and added by using the actions **New > File > Advanced** or **New > Folder > Advanced** from the contextual menu or **File** menu. Also, the contextual menu action **Delete** can be used to remove them from the project (the local file system remains unchanged).

![Figure 24: The Navigator View with Examples of the Two Types of Resources](image)

**Creating New Projects**

The following actions are available by selecting **New** from the contextual menu or **File** menu:

- **New > XML Project**
  Opens the **New XML Project** dialog that allows you to create a new project and adds it to the project structure in the **Navigator** view.

- **New > Sample XML Project**
  Opens the **New sample XML project** dialog that allows you to customize sample resources in a new project and adds it to the project structure in the **Navigator** view.

- **New > New from Templates**
  Creates a new file from a list of templates that are included in Oxygen XML Developer plugin and adds the new file into the selected folder in the project structure.
To create a new XML project, follow these steps:

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

Managing Project Content
Creating/Adding Files and Folders

You can create linked folders (shortcuts) by dragging and dropping folders from the Windows Explorer / Mac OS X Finder to the project tree, or by using the contextual menu from the location in the project tree where you wanted it added and selecting New > Folder > Advanced. To create a file inside a linked folder, use the contextual menu and select New > File (you can use the Advanced button to link to a file in the local file system).

Note: The linked folders presented in the Navigator view are marked with a special icon.

You can create physical folders by selecting New > Folder from the contextual menu.

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

Removing Files and Folders

To remove one or more files or folders, select them in the project tree and press the Delete key, or select the contextual menu action ☠ Delete.

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

Moving Files and Folders

You can move the resources of the project with drag and drop operations on the files and folders of the tree.

You can also use the usual Copy and Paste actions to move resources in the Navigator view.

Renaming Files and Folders

There are two ways you can rename an item in the Navigator view. Select the item in the Navigator view and do one of the following:

• Invoke the Rename action from the contextual menu.
• Press F2 and type the new name.
  To finish editing the item name, press Enter.

Locating and Opening Files

If a project folder contains a lot of documents, a certain document can be located quickly in the project tree by selecting the folder containing the desired document and typing the first few characters of the document name. The desired document is automatically selected as soon as the typed characters uniquely identify its name in the folder.

The selected document can be opened by pressing the Enter key, by double-clicking it, or with one of the Open actions from the contextual menu. The files with known document types are opened in the associated editor, while binary files are opened with the associated system application. To open a file with a known document type in an editor other than the default one, use the Open with action. Also, dragging and dropping files from the project tree to the editor area results in the files being opened.
Saving the Project

The project file is automatically saved every time the content of the Project view is saved or modified by actions such as adding or removing files and drag and drop.

Validate Files

The currently selected files associated with the Oxygen XML Developer plugin in the Package Explorer view or in the Navigator view can be checked to be XML well-formed or validated against a schema (DTD, XML Schema, Relax NG, Schematron or NVDL) with one of the following contextual menu actions found in the Validate sub-menu:

- **Check Well-Formedness**
  Checks if the selected file or files are well-formed.

- **Validate**
  Validates the selected file or files against their associated schema. EPUB files make an exception, because this action triggers a Validate and Check for Completeness operation.

- **Validate with Schema...**
  Validates the selected file of files against a specified schema.

- **Configure Validation Scenario(s)...**
  Allows you to configure and run a validation scenario.

Applying Transformation Scenarios

The currently selected files associated with the Oxygen XML Developer plugin in the Package Explorer view or in the Navigator view can be transformed in one step with one of the following actions available from contextual menu in the Transform sub-menu:

- **Transform > Play Apply Transformation Scenario(s)**
  Obtains the output with one of the built-in scenarios.

- **Transform > Edit Configure Transformation Scenario(s)...**
  Opens a dialog that allows you to configure pre-defined transformation scenarios.

- **Transform > Move Transform...**
  Allows you to select a transformation scenario to be applied to the currently selected files.

Other Contextual Menu Actions

Other actions that are available in the contextual menu from the project tree include:

- **Open**
  Displays the Open file dialog.

- **Open with submenu**
  This submenu offers you choices for opening the selected file in various editors.

- **Refactoring > Rename resource...** (Available for certain document types (such as XML, XSD, and XSL files)
  Allows you to change the name of a resource.

- **Refactoring > Move resource...** (Available for certain document types (such as XML, XSD, and XSL files)
  Allows you to change the location on disk of a resource.

- **Refactoring > XML Refactoring...**
  Opens the XML Refactoring tool wizard that presents refactoring operations to assist you with managing the structure of your XML documents.
**Refresh**
Refreshes the content and the dependencies between the resources in the *Master Files* directory.

**// XPath in Files...**
Opens the *XPath/XQuery Builder* view that allows you to compose XPath and XQuery expressions and execute them over the currently edited XML document.

**✓ Check Spelling in Files...**
Allows you to check the spelling of multiple files.

**Properties**
Displays the properties of the current file in a *Properties* dialog box.

---

**Moving/Renaming Resources in the Navigator View**
The *Navigator* view allows you to move or rename a file from the current project.

To move a file or a directory, drag and drop it to the new location in the tree structure from the *Navigator* view. You can also use the usual Copy and Paste actions or right click the file or directory and select the Move... action from its contextual menu. Oxygen XML Developer plugin presents a Move Resources dialog box that allows you to choose its destination.

To quickly rename a file or a directory, use the in-place editing either by pressing F2 or by selecting Rename from the contextual menu of the resource.

**Problems with Updating References of Moved/Renamed Resources**
In some case the references of a moved or a renamed resource can not be updated. For example, when a resource is resolved through an XML catalog or when the path to the moved or renamed resource contains entities. For these cases, Oxygen XML Developer plugin displays a warning dialog.
Figure 25: Problems Dialog

Defining Master Files at Project Level

This chapter details the Master Files Support available in Oxygen XML Developer plugin.

The Master Files Support helps you simplify the configuration and development of XML projects. A Master File typically refers to the root of an import/include tree of modules.

Introduction

Oxygen XML Developer plugin allows you to define master files at project level. These master files are automatically used by Oxygen XML Developer plugin to determine the context for operations such as validation, content completion, refactoring, or search for XML, XSD, XSL, WSDL, and RNG modules. Oxygen XML Developer plugin maintains the hierarchy of the master files, helping you to determine the editing context.

To watch our video demonstration about the Master Files Support for XML documents, XSL documents, and WSDL documents, go to Working with Modular XML Files, Master Files Support, and Working with Modular WSDL Files, respectively.

Master Files Benefits

When you edit a module after defining the master files, you have the following benefits:

• When the module is validated, Oxygen XML Developer plugin automatically identifies the master files that include the module and validates all of them.
• The Content Completion Assistant presents all the components that are collected, from the master files to the modules they include.
The Outline view displays all the components that are defined in the master files hierarchy.

The master files that are defined for the current module determines the scope of the search and refactoring actions. Oxygen XML Developer plugin performs the search and refactoring actions in the context that the master files determine, thus improving the speed of execution.

### Enabling the Master Files Support

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

### Detecting Master Files

Oxygen XML Developer plugin allows you to detect the master files using the Detect Master Files... option available in the contextual menu of the project. This action applies to the folders you select in the project. To detect master files over the entire project, do one of the following:

- Right-click the root of the project and select Detect Master Files....
- Use the Detect Master Files from Project... option, available in the contextual menu of the Master Files folder.

Both of these options display the Detect Master Files wizard. In the first panel you can select the type of master files you want Oxygen XML Developer plugin to detect. In the subsequent panel the detected master files are presented in a tree-like fashion. The resources are grouped into three categories:

- Possible master files - the files presented on the first level in this category are not imported/included from other files. These files are most likely to be set as master files.
- Cycles - the files that are presented on the first level have circular dependencies between them. Any of the files presented on the first level of a cycle is a possible master file.
- Standalone - files that do not include/import other files and are also not included/imported themselves. It is not necessary to set them as master files.

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

The master files that are already defined in the project are automatically marked in the tree and cannot be removed. The only way to disable a master file is to delete it from the Master Files folder.

The next panel displays a list with the selected master files. Click the Finish button to add the master files in the Master Files folder.

You can use the Select Master Files option to automatically mark all master files. This action sets all the resources from the Possible Master Files category and the first resource of each Cycle as master files.

**Tip:** We recommend you to only add top-level files (files that are at the root of the include/import graph) in the Master Files directory. Keep the file set to a minimum and only add files that import or include other files.

### Adding/Removing a Master File

The Master Files directory only contains logical folders and linked files. To add files in the Master Files directory, use one of the following methods:

- Right-click a file from your project and select Add to Master Files from the contextual menu.
- Drag and drop files into the Master Files directory.
- From the contextual menu of the Resource Hierarchy Dependencies view, use the Add to Master Files action.
You can view the *master files* for the currently edited resource in the Editor Properties view.

## Editing XML Documents

This section explains the XML editing features of the application. All the user interface components and actions available to users are described in detail with appropriate procedures for various tasks.

### Associate a Schema to a Document

This section explains the methods of associating a schema to a document for validation and content completion purposes.

#### Setting a Schema for Content Completion

This section explains the available methods of setting a schema for content completion in an XML document edited in Oxygen XML Developer plugin.

### Supported Schema Types for XML Documents

The supported schema types are:

- W3C XML Schema 1.0 and 1.1 (with and without embedded Schematron rules);
- DTD;
- Relax NG - XML syntax (with and without embedded Schematron rules);
- Relax NG - compact syntax;
- NVDL;
- Schematron (both ISO Schematron and Schematron 1.5).

#### Setting a Default Schema

When trying to detect a schema, Oxygen XML Developer plugin searches in multiple locations, in the exact following order:

- The *validation scenario* associated with the document.
- The validation scenario associated with the document type (if defined).
- The document schema declaration.
- The document type schema definition. Each document type available in Document Type Association preferences page contains a set of rules for associating a schema with the current document.

Note: The locations are sorted by priority, from high to low.

The schema has one of the following types: XML Schema, XML Schema with embedded Schematron rules, Relax NG (XML syntax or compact syntax), Relax NG (XML syntax) with embedded Schematron rules, Schematron, DTD, NVDL.

The rules are applied in the order they appear in the table and take into account the local name of the root element, the default namespace and the file name of the document.

Important:

The editor is creating the content completion lists by analysing the specified schema and the current context (the position in the editor). If you change the schema, then the list of tags to be inserted is updated.
Making the Schema Association Explicit in the XML Instance Document

The schema used by the Content Completion Assistant and document validation engine can be associated with the document using the Associate Schema action. For most of the schema types, it uses the xml-model processing instruction, the exceptions being:

- W3C XML Schema - the xsi:schemaLocation attribute is used.
- DTD - the DOCTYPE declaration is used.

The association can specify a relative file path or a URL of the schema. The advantage of relative file path is that you can configure the schema at file level instead of document type level.

Select the Associate schema action from the Document > Schema menu or the Document toolbar to select the schema that will be associated with the XML document. The Associate Schema dialog box is displayed:

![Associate Schema Dialog Box](image)

**Figure 27: The Associate Schema Dialog Box**

The available options are:

- **URL** - Contains a predefined set of schemas that are used more often and it also keeps a history of the last used schemas. The URL must point to the schema file which can be loaded from the local disk or from a remote server through HTTP(S), FTP(S).
• **Schema type** - Selected automatically from the list of possible types in the Schema type combo box (XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, NVDL) based on the extension of the schema file that was entered in the URL field.

• **Public ID** - Specify a public ID if you have selected a DTD.

• **Add additional association for embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules, enable this option.

• **Use path relative to file location** - Enable this option if the XML instance document and the associated schema contain relative paths. The location of the schema file is inserted in the XML instance document as a relative file path. This practice allows you, for example, to share these documents with other users, without running into problems caused by different project locations on physical disk.

• **Keep existing schema associations** - Enable this option to keep the associations of the currently edited document with a Schema when you associate a new one.

The association with an XML Schema is added as an attribute of the root element. The **Associate schema** action adds:

- a `xsi:schemaLocation` attribute, if the root element of the document sets a default namespace with an `xmlns` attribute.
- or a `xsi:noNamespaceSchemaLocation` attribute, if the root element does not set a default namespace.

The association with a DTD is added as a `DOCTYPE` declaration. The association with a Relax NG, Schematron or NVDL schema is added as **xml-model processing instruction**.

**Associating a Schema With the Namespace of the Root Element**

The namespace of the root element of an XML document can be associated with an XML Schema using an **XML catalog**. If there is no `xsi:schemaLocation` attribute on the root element and the XML document is not matched with a document type, the namespace of the root element is searched in the **XML catalogs set in Preferences**.

If the XML catalog contains an `uri` or `rewriteUri` or `delegateUri` element, its schema will be used by the application to drive the **content completion** and document **validation**.

**The xml-model Processing Instruction**

The `xml-model` processing instruction associates a schema with the XML document that contains the processing instruction. It must be added at the beginning of the document, just after the XML prologue. The following code snippet contains an `xml-model` processing instruction declaration:

```xml
<?xml-model href="../schema.sch" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron" phase="ALL" title="Main schema"/>
```

It is available in the **Content Completion Assistant**, before XML document root element, and includes the following attributes:

- `href` (required) - The schema file location.
- `type` - The content type of the schema. This is an optional attribute with the following possible values for each specified type:
  - DTD - The recommended value is `application/xml-dtd`.
  - W3C XML Schema - The recommended value is `application/xml`, or can be left unspecified.
  - RELAX NG XML Syntax - The recommended value is `application/xml`, or can be left unspecified.
  - RELAX NG Compact Syntax - The recommended value is `application/relax-ng-compact-syntax`.
  - Schematron - The recommended value is `application/xml`, or can be left unspecified.
  - NVDL - The recommended value is `application/xml`, or can be left unspecified.

- `schematypens` - The namespace for the schema language of the referenced schema with the following possible values:
  - DTD - Not specified.
  - RELAX NG XML Syntax - The recommended value is `http://relaxng.org/ns/structure/1.0`.
• RELAX NG Compact Syntax - Not specified.
• Schematron - The recommended value is http://purl.oclc.org/dsdl/schematron.
• NVDL - The recommended value is http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0.

• phase - The phase name for the validation function in Schematron schema. This is an optional attribute. To run all phases from the Schematron, use the special #ALL value. If the phase is not specified, the default phase that is configured in the Schematron will be applied.
• title - The title for the associated schema. This is an optional attribute.

Older versions of Oxygen XML Developer plugin used the oxygen processing instruction with the following attributes:
• RNGSchema - Specifies the path to the Relax NG schema that is associated with the current document.
• type - Specifies the type of Relax NG schema. It is used along with the RNGSchema attribute and can have the value xml or compact.
• NVDLSchema - Specifies the path to the NVDL schema that is associated with the current document.
• SCHSchema - Specifies the path to the SCH schema that is associated with the current document.

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

Learning Document Structure

When working with documents that do not specify a schema, or for which the schema is not known or does not exist, Oxygen XML Developer plugin is able to learn and translate the document structure to a DTD. You can choose to save the learned structure to a file in order to provide a DTD as an initialization source for content completion and document validation. This feature is also useful for producing DTD's for documents containing personal or custom element types.

When you open a document that is not associated with a schema, Oxygen XML Developer plugin automatically learns the document structure and uses it for content completion. To disable this feature you have to uncheck the checkbox Learn on open document in the user preferences.

Create a DTD from Learned Document Structure

When there is no schema associated with an XML document, Oxygen XML Developer plugin can learn the document structure by parsing the document internally. This feature is enabled with the option Learn on open document that is available in the user preferences.

To create a DTD from the learned structure:

1. Open the XML document for which a DTD will be created.
2. Go to XML > Learn Structure > Ctrl Shift L (Command Shift L on OS X).
   The Learn Structure action reads the mark-up structure of the current document. The Learn completed message is displayed in the application’s status bar when the action is finished.
3. Go to XML > Save Structure > Ctrl Shift S (Command Shift S on OS X). Enter the DTD file path.
4. Press the Save button.

Content Completion Assistant

The intelligent Content Completion Assistant available in Oxygen XML Developer plugin enables rapid, in-line identification and insertion of structured language elements, attributes and, in some cases, their parameter options.
The functioning of the Content Completion Assistant feature is schema-driven (XML Schema, DTD, and RELAX NG). When Oxygen XML Developer plugin detects a schema, it logs its URL in the Status view.

The **Content Completion Assistant** is enabled by default. To disable it, open the Preferences dialog box and go to Editor > Content Completion. It is activated:

- automatically, after a configurable delay from the last key press of the `<` character. You can adjust the delay from the Content Completion preferences page
- on demand, by pressing **Ctrl Space** (**Command Space on OS X**) on a partial element or attribute name.

**Note:** If the Content Completion list contains only one valid proposal, when you press the **Ctrl Space** (**Command Space on OS X**) shortcut key, the proposal is automatically inserted.

When active, it displays a list of context-sensitive proposals valid at the current caret position. Elements are highlighted in the list using the Up and Down cursor keys on your keyboard. For each selected item in the list, the **Content Completion Assistant** displays a documentation window. You can customize the size of the documentation window by dragging its top, right, and bottom borders.

To insert the selected content:

- press Enter or Tab on your keyboard to insert both the start and end tags.
- press **Ctrl Enter** (**Command Enter on OS X**) on your keyboard. Oxygen XML Developer plugin inserts both the start and end tags and positions the cursor between the tags, so you can start typing content.

**Note:** When the DTD, XML Schema or RELAX NG schema specifies required child elements for the newly added element, they are inserted automatically only if the Add Element Content option (found in the Content Completion preferences page) is enabled. The **Content Completion Assistant** can also add optional content and first choice particle, as specified in the DTD or XML Schema or RELAX NG schema. To activate this feature, open the Preferences dialog box, go to **Content Completion**, and select the **Add optional content** and **Add first Choice particle** check boxes.

After inserting an element, the cursor is positioned:

- before the `>` character of the start tag, if the element allows attributes, in order to enable rapid insertion of any of the attributes supported by the element. Pressing the space bar displays the Content Completion list once again. This time it contains the list of allowed attribute names. If the attribute supports a fixed set of parameters, the assistant list displays the list of valid parameters. If the parameter setting is user-defined and therefore variable, the assistant is closed to enable manual insertion. The values of the attributes can be learned from the same elements in the current document.
- after the `>` character of the start tag if the element has no attributes.

The **Content Completion Assistant** is displayed:
• anywhere within a tag name or at the beginning of a tag name in an XML document, XML Schema, DTD, or Relax NG (full or compact syntax) schema
• anywhere within an attribute name or at the beginning of an attribute name in any XML document with an associated schema
• within attribute values or at the beginning of attribute values in XML documents where lists of possible values have been defined for that element in the schema associated with the document.

The items that populate the **Content Completion Assistant** depend on the element structure specified in the DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema associated with the edited document.

**Note:** The **Content Completion Assistant** is able to offer elements defined both by XML Schemas version 1.0 and 1.1.

The number and type of elements displayed by the **Content Completion Assistant** is dependent on the cursor’s current position in the structured document. The child elements displayed within a given element are defined by the structure of the specified DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema.

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

A schema may declare certain attributes as **ID** or **IDREF**/**IDREFS**. When the document is validated, Oxygen XML Developer plugin checks the uniqueness and correctness of the ID attributes. It also collects the attribute values declared in the document to prepare the **Content Completion Assistant**’s list of proposals. This is available for documents that use DTD, XML Schema, and Relax NG schema.

Also, values of all the **xml:id** attributes are handled as ID attributes. They are collected and displayed by the **Content Completion Assistant** as possible values for **anyURI** attributes defined in the schema of the edited document. This works only for XML Schema and Relax NG schemas.

For documents that use an XML Schema or Relax NG schema, the content assistant offers proposals for attributes and elements values that have as type an enumeration of tokens. Also, if a default value or a fixed value is defined in the XML Schema used in validation for an attribute or element, then that value is offered in the **Content Completion Assistant** window.

**Set Schema for Content Completion**

The DTD, XML Schema, Relax NG, or NVDL schema used to populate the **Content Completion Assistant** is specified in the following methods, in order of precedence:

• The schema specified explicitly in the document. In this case Oxygen XML Developer plugin reads the beginning of the document and resolves the location of the DTD, XML Schema, Relax NG schema, or NVDL schema.
• The default schema rule declared in the **Document Type Association** preferences panel which matches the edited document.
• For XSLT stylesheets, the schema specified in the Oxygen XML Developer plugin **Content Completion options**. Oxygen XML Developer plugin will read the Content Completion settings when the prolog fails to provide or resolve the location of a DTD, XML Schema, Relax NG or NVDL schema.
• For XML Schemas, the schema specified in the Oxygen XML Developer plugin **Content Completion options**. Oxygen XML Developer plugin will read the Content Completion settings and the specified schema will enhance the content completion inside the **xs:annotation/xs:appinfo** elements of the XML Schema.

**Content Completion in Documents with Relax NG Schemas**

Inside the documents that use a Relax NG schema, the **Content Completion Assistant** is able to present element values if such values are specified in the Relax NG schema. Also in Relax NG documents the **Content Completion Assistant** presents additional values of type ID for an **anyURI** data type. It presents also pattern names defined in the Relax NG schema as possible values for pattern references. For example if the schema defines an **enumValuesElem** element like:

```
<element name="enumValuesElem">
  <choice>
```

In documents based on this schema, the **Content Completion Assistant** offers the following list of values:

```
<value>value1</value>
<value>value2</value>
<value>value3</value>
</choice>
</element>
```

**Figure 29: Content Completion assistant - element values in Relax NG documents**

**Schema Annotations**

A schema annotation is a documentation snippet associated with the definition of an element or attribute in a schema. If such a schema is associated with an XML document, the annotations are displayed in:

- the **Content Completion Assistant**.
- a small tooltip window shown when the mouse hovers over an element or attribute.

The schema annotations support is available if the schema type is one of the following: XML Schema, Relax NG, NVDL, or DTD. If you want to turn off this feature, disable the **Show annotations in Content Completion Assistant** option.

**Styling Annotations with HTML**

You can use HTML format in the annotations you add in an XML Schema or Relax NG schema. This improves the visual appearance and readability of the documentation window displayed when editing XML documents validated against such a schema. An annotation is recognized and displayed as HTML if it contains at least one HTML element, like: `div`, `body`, `p`, `br`, `table`, `ul`, or `ol`.

The HTML rendering is controlled by the **Show annotations using HTML format, if possible** option. When this option is disabled, the annotations are converted and displayed as plain text. If the annotation contains one or more HTML tags (`p`, `br`, `ul`, `li`), they are rendered as an HTML document loaded in a web browser: `p` begins a new paragraph, `br` breaks the current line, `ul` encloses a list of items, `li` encloses an item of the list.

**Collecting Annotations from XML Schemas**

In an XML Schema the annotations are specified in an `<xs:annotation>` element like this:

```
<xs:annotation>
  <xs:documentation>
    Description of the element.
  </xs:documentation>
</xs:annotation>
```

For XML Schema, if an element or attribute does not have a specific annotation, then Oxygen XML Developer plugin looks for an annotation in the type definition of that element or attribute.

**Collecting Annotations from Relax NG Schemas**

For Relax NG schema element / attribute annotation are made using the `<documentation>` element from the [http://relaxng.org/ns/compatibility/annotations/1.0 namespace](http://relaxng.org/ns/compatibility/annotations/1.0). However, any element outside the Relax NG namespace ([http://relaxng.org/ns/structure/1.0](http://relaxng.org/ns/structure/1.0)) is handled as annotation and the text content is displayed in the annotation window. To activate this behaviour, enable the **Use all Relax NG annotations as documentation** option.
Collecting Annotation from DTDs

For DTD Oxygen XML Developer plugin defines a custom mechanism for annotation using comments enabled from the option *Use DTD comments as annotations*. Following is an example of a DTD annotation:

```xml
<!--doc:Description of the element. -->
```

Content Completion Helper Views

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

The Model View

The **Model** view presents the structure of the currently edited tag and tag documentation defined as annotation in the schema of the current document. Open the **Model** view from **Window > Show View > Other > oXygen XML Editor > Model view**

![Model View](image)

**Figure 30:** The Model View

*The Element Structure Panel*

The element structure panel shows the structure of the current edited or selected tag in a tree-like format. The information includes the name, model and attributes the currently edited tag may have. The allowed attributes are shown along with imposed restrictions, if any.
The Annotation Panel

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

The Attributes View

The Attributes View presents all possible attributes of the current element.

The attributes present in the document are rendered bold in the Attributes View. You can start editing the value of an attribute by clicking the Value cell of a table row. If the possible values of the attribute are specified as list in the schema associated with the edited document, the Value cell works as a list box from which you can select one of the possible values to be inserted in the document.

The Attributes table is sortable, three sorting modes being available by clicking the Attribute column name: alphabetically ascending, alphabetically descending, or custom order. The custom order places the already used attributes at the beginning of the table, as they appear in the element, followed by the rest of the allowed elements, as they are declared in the associated schema.
The Elements View

The Elements view presents a list of all defined elements that you can insert at the current caret position according to the schema associated to the document. Double-clicking any of the listed elements inserts that element in the edited document. All elements from a sequence are presented but the invalid proposals (which cannot be inserted in the current context) are grayed-out.

The Entities View

This view displays a list with all entities declared in the current document, as well as built-in ones. Double-clicking one of the entities will insert it at the current cursor position. You can also sort entities by name and value by clicking the column headers.
Figure 35: The Entities View

The view features a filtering capability that allows you to search an entity by name, value, or both. Also, you can choose to display the internal or external entities.

Note: When entering filters, you can use the ? and * wildcards. Also, you can enter multiple filters by separating them with comma.

Code Templates

Code templates are code fragments that can be inserted quickly at the current editing position. Oxygen XML Developer plugin comes with a set of built-in code templates for CSS, LESS, Schematron, XSL, XQuery, and XML Schema document types. You can also define your own code templates and share them with others.

To get a complete list of available code templates, press Ctrl Shift Space (Command Shift Space on OS X) in Text mode or Enter in Author mode. To enter the code template, select it from the list or type its code and press Enter. If a shortcut key has been assigned to the code template, you can also use the shortcut key to enter it. Code templates are displayed with a .tl symbol in the content completion list.

When the Content Completion Assistant is invoked (Ctrl Space (Command Space on OS X)), it also presents a list of code templates specific to the type of the active editor.

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

Configuring the Proposals in the Content Completion Assistant

Oxygen XML Developer plugin gathers information from the associated schemas (DTDs, XML Schema, RelaxNG) to determine the proposals that appear in the Content Completion Assistant. Oxygen XML Developer plugin also includes support that allows you to configure the possible attribute or element values for the proposals. To do so, a configuration file can be used, along with the associated schema, to add or replace possible values for attributes or elements that are proposed in the Content Completion Assistant. An example of a specific use-case is if you want the Content Completion Assistant to propose several possible values for the language code whenever you use an xml:lang attribute.

To configure content completion proposals, follow these steps:

1. Create a new resources folder (if it does not already exist) in the frameworks directory for the document type. For instance: OXYGEN_INSTALL_DIR/frameworks/dita/resources.
2. Open the Preferences dialog box and go to Document Type Association. Edit the document type configuration for your XML vocabulary, and in the Classpath tab add a link to that resources folder.
3. Use the **New** document wizard to create a configuration file using the **Content Completion Configuration** file template.

4. Make the appropriate changes to your custom configuration file. The file template includes details about how each element and attribute is used in the configuration file.

5. Save the file in the `resources` folder, using the fixed name: `cc_value_config.xml`.

6. Re-open the application and open an XML document. In the **Content Completion Assistant** you should see your customizations.

**The Configuration File**

The configuration file is composed of a series of `match` instructions that will match either an element or an attribute name. A new value is specified inside one or more `item` elements, which are grouped inside an `items` element. The behavior of the `items` element is specified with the help of the `action` attribute, which can have any of the following values:

- **append** - Adds new values to appear in the proposals list (default value).
- **addIfEmpty** - Adds new values to the proposals list, only if no other values are contributed by the schema.
- **replace** - Replaces the values contributed by the schema with new values to appear in the proposals list.

The values in the configuration file can be specified either directly or by calling an external XSLT file that will extract data from any external source.

**Example - Specifying Values Directly**

```xml
<match elementName="lg" elementNS="http://www.oxygenxml.com/ns/samples">
  <items action="replace">
    <item value="stanza"/>
    <item value="refrain"/>
  </items>
</match>

<match attributeName="type">
  <items>
    <item value="stanza"/>
    <item value="refrain"/>
  </items>
</match>
```

**Example - Calling an External XSLT Script**

```xml
<xslt href="../xsl/get_values_from_db.xsl" useCache="false" action="replace"/>
```

In this example, the `get_values_from_db.xsl` is executed in order to extract values from a database.

>Note: A comprehensive XSLT sample is included in the **Content Completion Configuration** file template.

**Validating XML Documents**

The W3C XML specification states that a program should not continue to process an XML document if it finds a validation error. The reason is that XML software should be easy to write, and that all XML documents should be compatible. With HTML it was possible to create documents with lots of errors (like when you forget an end tag). One of the main reasons that HTML browsers are so big and incompatible, is that they have their own ways to figure out what a document should look like when they encounter an HTML error. With XML this should not be possible.

However, when creating an XML document, errors are very easily introduced. When working with large projects or many files, the probability that errors will occur is even greater. Determining that your project is error-free can be time consuming and even frustrating. For this reason Oxygen XML Developer plugin provides functions that enable easy error identification and rapid error location.
Checking XML Well-formedness

A Well-Formed XML document is a document that conforms to the XML syntax rules. A Namespace Well-Formed XML document is a document that is XML Well-Formed and is also namespace-wellformed and namespace-valid.

The XML Syntax rules for Well-Formed XML are:

- All XML elements must have a closing tag.
- XML tags are case-sensitive.
- All XML elements must be properly nested.
- All XML documents must have a root element.
- Attribute values must always be quoted.
- With XML, white space is preserved.

The namespace-wellformed rules are:

- All element and attribute names contain either zero or one colon.
- No entity names, processing instruction targets, or notation names contain any colons.

The namespace-valid rules are:

- The prefix xml is by definition bound to the namespace name http://www.w3.org/XML/1998/namespace. It MAY, but need not, be declared, and MUST NOT be undeclared or bound to any other namespace name. Other prefixes MUST NOT be bound to this namespace name.
- The prefix xmlns is used only to declare namespace bindings and is by definition bound to the namespace name http://www.w3.org/2000/xmlns/. It MUST NOT be declared or undeclared. Other prefixes MUST NOT be bound to this namespace name.
- All other prefixes beginning with the three-letter sequence x, m, l, in any case combination, are reserved. This means that users SHOULD NOT use them except as defined by later specifications and processors MUST NOT treat them as fatal errors.
- The namespace prefix, unless it is xml or xmlns, MUST have been declared in a namespace declaration attribute in either the start-tag of the element where the prefix is used or in an ancestor element (i.e. an element in whose content the prefixed markup occurs). Furthermore, the attribute value in the innermost such declaration MUST NOT be an empty string.

To check if a document is Namespace Well-Formed XML, select the Check Well-Formedness action from the XML menu or from the Validation toolbar drop-down list. If any error is found the result is returned to the message panel. Each error is one record in the result list and is accompanied by an error message. Clicking the record will open the document containing the error and highlight its approximate location.

A not Well-Formed XML Document

```xml
<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

```xml
<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

When Check document form is performed the following error is raised:
The prefix "x" for element "x:y" is not bound.

Also the selected files in the current project can be checked for well-formedness with a single action by selecting the ✔ Check Well-Formedness action from the Validate submenu when invoking the contextual menu in the Navigator view.

Validating XML Documents Against a Schema

A Valid XML document is a Well-Formed XML document that also conforms to the rules of a schema that defines the legal elements of an XML document. The schema type can be: XML Schema, Relax NG (full or compact syntax), Schematron, Document Type Definition (DTD), or Namespace-based Validation Dispatching Language (NVDL).

The purpose of the schema is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

The ✔ Validate function ensures that your document is compliant with the rules defined by an associated DTD, XML Schema, Relax NG, or Schematron schema. XML Schema or Relax NG Schema can embed Schematron rules. For Schematron validations you can select the validation phase.

Marking Validation Errors and Warnings

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

The ruter 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 ruter contains the following areas:

• Top area containing a success validation indicator that will turn green in case the validation succeeded or red otherwise.
• Middle area where the error markers are depicted in red. To limit the number of markers shown open the Preferences dialog box and go to Editor > Document checking > Maximum number of problems reported per document.

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

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

If you want to see all the validation error messages grouped in a view you should use the ✔ Validate action from the XML menu or from the ✔ Validation toolbar drop-down list. This action collects all error messages in the Problems view of the Eclipse platform if the validated file is in the current workspace or in a custom Oxygen view called Errors if the validated file is outside the workspace.

Customising Assert Error Messages

To customise the error messages that the Xerces or Saxon validation engines display for the assert and assertion elements, set the message attribute on these elements. For Xerces, the message attribute has to belong to the http://xerces.apache.org namespace. For Saxon, the message attribute has to belong to the http://saxon.sourceforge.net/ namespace. The value of the message attribute is the error message displayed in case the assertion fails.
Validation Example - A DocBook Validation Error

In the following DocBook 4 document the content of the listitem element does not match the rules of the DocBook 4 schema, that is docbookx.dtd.

```xml
<?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|classsynopsis|constructionsynopsis|destructorsynopsis|methodsynopsis
|formalpara|para|simpara|address|blockquote|graphic|graphicco|mediaobject|mediaobjectco
|informalequation|informalexample|informalfigure|informaltab|equation|example|figure
|table|msgset|procedure|sidebar|qandaset|task|anchor|bridgehead|remark|highlights
|abstract|author|blurb|epigraph|indexterm|beginpage)+".
```

This error message is a little more difficult to understand, so understanding of the syntax or processing rules for the DocBook XML DTD's listitem element is recommended. However, the error message does give us a clue as to the source of the problem, indicating that “The content of element type c must match”.

Luckily most standards based DTD's, XML Schema's and Relax NG schemas are supplied with reference documentation. This enables us to lookup the element and read about it. In this case you should learn about the child elements of listitem and their nesting rules. Once you have correctly inserted the required child element and nested it in accordance with the XML rules, the document will become valid on the next validation test.

**Automatic Validation**

Oxygen XML Developer plugin *can be configured* to mark validation errors in the document as you are editing. If you *enable the Automatic validation option* any validation errors and warnings will be *highlighted automatically in the editor panel*. The automatic validation starts parsing the document and marking the errors after a *configurable delay* from the last key typed. Errors are highlighted with underline markers in the main editor panel and small rectangles on the right side ruler of the editor panel, in the same way as for manual validation invoked by the user.
Custom Validators

If you need to validate the edited document with a validation engine that is different from the built-in engine, you can configure external validators in the Oxygen XML Developer plugin preferences. After a custom validation engine is properly configured, it can be applied on the current document by selecting it from the list of custom validation engines in the Validation toolbar drop-down list. The document is validated against the schema declared in the document.

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

- **LIBXML** - Included in Oxygen XML Developer plugin (Windows edition only). It is associated to XML Editor. It is able to validate the edited document against XML Schema, Relax NG schema full syntax, internal DTD (included in the XML document) or a custom schema type. XML catalogs support (the --catalogs parameter) and XInclude processing (--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, add the --dtdvalid ${ds} parameter manually to the DTD validation command line. ${ds} represents the detected DTD declaration in the XML document.

**Caution:** File paths containing spaces are not handled correctly in the LIBXML processor. For example the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled by LIBXML if Oxygen XML Developer plugin is installed in the default location on Windows (C:\Program Files) because the built-in XML catalog files are stored in the frameworks subfolder of the installation folder which in this case contains at least one space character in the file path.

**Attention:** On OS X if the full path to the LIBXML executable file is not specified in the Executable path text field, some errors may occur during validation against a W3C XML Schema, such as:

```
Unimplemented block at ... xmlschema.c
```

To avoid these errors, specify the full path to the LIBXML executable file.

- **Saxon SA** - Included in Oxygen XML Developer plugin. It is associated to XML Editor and XSD Editor. It is able to validate XML Schema schemas and XML documents against XML Schema schemas. The validation is done according to the W3C XML Schema 1.0 or 1.0. This can be configured in Preferences.

- **MSXML 4.0** - Included in Oxygen XML Developer plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
• **MSXML.NET** - Included in Oxygen XML Developer plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.

• **XSV** - Not included in Oxygen XML Developer plugin. Windows and Linux distributions of XSV can be downloaded from http://www.cogsci.ed.ac.uk/~ht/xsv-status.html. The executable path is already configured in Oxygen XML Developer plugin for the [OXYGEN_DIR]/xsv installation folder. If it is installed in a different folder the predefined executable path must be corrected in Preferences. It is associated to XML Editor and XSD Editor. It is able to validate the edited document against XML Schema or a custom schema type.

• **SQC (Schema Quality Checker from IBM)** - Not included in Oxygen XML Developer plugin. It can be downloaded from here (it comes as a .zip file, at the time of this writing SQC2.2.1.zip is about 3 megabytes). The executable path and working directory are already configured for the SQC installation directory [OXYGEN_DIR]/sqc. If it is installed in a different folder the predefined executable path and working directory must be corrected in the Preferences page. It is associated to XSD Editor.

**Linked Output Messages of an External Engine**

Validation engines display messages in an output view at the bottom of the Oxygen XML Developer plugin window. If such an output message (warning, error, fatal error, etc) spans between three to six lines of text and has the following format, then the message is linked to a location in the validated document. A click on the message in the output view highlights the location of the message in an editor panel containing the file referenced in the message. This behavior is similar to the linked messages generated by the default built-in validator.

Linked messages have the following format:

- **Type:** the string `Type:` followed by a letter for the type of the message: fatal error, error, warning - this property is optional in a linked message
- **SystemID:** a system ID of a file (the string `SystemID:` followed by the system ID of the file that will be opened for highlighting when the message is clicked in the output message - usually the validated file, the schema file or an included file)
- **Line:** a line number (the string `Line:` followed by the number of the line that will be highlighted)
- **Column:** a column number (the string `Column:` followed by the number of the column where the highlight will start on the highlighted line) - this property is optional in a linked message
- **EndLine:** a line number (the string `EndLine:` followed by the number of the line where the highlight ends) - this property is optional in a linked message
- **EndColumn:** a column number (the string `EndColumn:` followed by the number of the column where the highlight ends on the end line) - this property is optional in a linked message

**Note:** The `Line/Column` pair works in conjunction with the `EndLine/EndColumn` pair. Thus, if both pairs are specified, then the highlight starts at `Line/Column` and ends at `EndLine/EndColumn`. If the `EndLine/EndColumn` pair is missing, the highlight starts from the beginning of the line identified by the `Line` parameter and ends at the column identified by the `Column` parameter.

- **AdditionalInfoURL:** the URL string pointing to a remote location where additional information about the error can be found - this line is optional in a linked message.
- **Description:** message content (the string `Description:` followed by the content of the message that will be displayed in the output view).

---

Example of how a custom validation engine can report an error using the format specified above:

<table>
<thead>
<tr>
<th>Type</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemID: file:///c:/path/to/validatedFile.xml</td>
<td></td>
</tr>
<tr>
<td>Line: 10</td>
<td></td>
</tr>
<tr>
<td>Column: 20</td>
<td></td>
</tr>
<tr>
<td>EndLine: 10</td>
<td></td>
</tr>
<tr>
<td>EndColumn: 35</td>
<td></td>
</tr>
<tr>
<td>AdditionalInfoURL: <a href="http://www.host.com/path/to/errors.html#errorID">http://www.host.com/path/to/errors.html#errorID</a></td>
<td></td>
</tr>
<tr>
<td>Description: custom validator message</td>
<td></td>
</tr>
</tbody>
</table>
Validation Scenario

A complex XML document is split in smaller interrelated modules. These modules do not make much sense individually and cannot be validated in isolation due to interdependencies with other modules. Oxygen XML Developer plugin validates the main module of the document when an imported module is checked for errors.

A typical example is the chunking DocBook XSL stylesheet which has chunk.xsl as the main module and param.xsl, chunk-common.xsl, and chunk-code.xsl as imported modules. param.xsl only defines XSLT parameters. The module chunk-common.xsl defines an XSLT template with the name chunk. Chunk-code.xsl calls this template. The parameters defined in param.xsl are used in the other modules without being redefined.

Validating chunk-code.xsl as an individual XSLT stylesheet, generates misleading errors in regards to parameters and templates that are used but undefined. These errors are only caused by ignoring the context in which this module is used in real XSLT transformations and in which it is validated. To validate such a module, define a validation scenario to set the main module of the stylesheet and the validation engine used to find the errors. Usually this engine applies the transformation during the validation process to detect the errors that the transformation generates.

You can validate a stylesheet with several engines to make sure that you can use it in different environments and have the same results. For example an XSLT stylesheet is applied with Saxon 6.5, Xalan and MSXML 4.0 in different production systems.

Other examples of documents which can benefit of a validation scenario are:

- A complex XQuery with a main module which imports modules developed independently but validated in the context of the main module of the query. In an XQuery validation scenario the default validator of Oxygen XML Developer plugin (Saxon 9) or any connection to a database that supports validation (Berkeley DB XML Database, eXist XML Database, Documentum xDb (X-Hive/DB) 10 XML Database, MarkLogic version 5 or newer) can be set as a validation engine.
- An XML document in which the master file includes smaller fragment files using XML entity references.

Note: When you validate a document for which a master file is defined, Oxygen XML Developer plugin uses the scenarios defined in the Master Files directory.

To watch our video demonstration about how to use a validation scenario in Oxygen XML Developer plugin, go to http://oxygenxml.com/demo/Validation_Scenario.html.

How to Create a Validation Scenario

Follow these steps for creating a validation scenario:

1. To open the Configure Validation Scenario dialog box, select the Configure Validation Scenario(s)... from the XML menu or the toolbar.
   The Configure Validation Scenario(s) dialog box is displayed. It contains the following types of scenarios:
   - Predefined scenarios are organized in categories depending on the type of file they apply to. You can identify Predefined scenarios by a yellow key icon that marks them as read-only. If the predefined scenario is the default scenario of the framework, its name is written in bold font. If you try to edit one of these scenarios, Oxygen XML Developer plugin creates a customizable duplicate.
   - User defined scenarios are organized under a single category, but you can use the drop-down option box to filter them by the type of file they validate.

   Note: The default validation scenarios are not displayed in the scenarios list. If the current file has no associated scenarios, the preview area displays a message to let you know that you can apply the default validation.
2. Press the **New** button to add a new scenario. The **New scenarios** dialog box that lists all validation units of the scenario is opened.

![Figure 37: Configure Validation Scenario](image)

The table includes the following information:

- **URL of the file to validate** - The URL of the main module that includes the current module. It is also the entry module of the validation process when the current one is validated.
- **File type** - The type of the document that is validated in the current validation unit. Oxygen XML Developer plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Developer plugin for validation of the type of document to which the current module belongs. **Default engine** is the default setting and it means that the default engine executes the validation. This engine is set in the **Preferences** pages for the current document type (XML document, XML Schema, XSLT stylesheet, XQuery file, etc.) instead of a validation scenario.
- **Automatic validation** - If this option is checked, the validation operation defined by this row is also applied by the automatic validation feature. If the **Automatic validation** feature is disabled in Preferences, then this option is ignored, as the Preference setting has a higher priority.

![Figure 38: Add / Edit a Validation Unit](image)
• **Schema** - This option becomes active when you set the **File type** to **XML Document**.

• **Settings** - Opens the Specify Schema dialog box that allows you to set a schema for validating XML documents, or a list of extensions for validating XSL or XQuery documents. You can also set a default phase for validation with a Schematron schema.

3. Press the **Add** button to add a new validation unit with default settings.

4. To edit the URL of the main validation module, double-click on its cell in the **URL of the file to validate** column. Specify the URL of the main module by doing one of the following:

   • Use the **Browse** drop-down button to browse for a local, remote, or archived file.

   • Use the **Insert Editor Variable** button to insert an **editor variable** or a **custom editor variable**.

   ```
   $(start-dir) - Start directory of custom validator
   $(standard-params) - list of standard parameters
   $(fn) - The current file name without extension
   $(currentFileURL) - The path of the currently edited file (URL)
   $(path) - The path of current file directory (URL)
   $(framework) - Oxygen frameworks directory (URL)
   $(root) - Project directory (URL)
   $(XInclude) - Oxygen installation directory (URL)
   $(home) - The path to user home directory (URL)
   $(cn) - Project name
   $(vsys(VAR_NAME)) - Value of environment variable VAR_NAME
   $(system(var.name)) - Value of system variable var.name
   ```

   **Figure 39: Insert an Editor Variable**

5. Select the type of the validated document.

   Note that this determines the list of possible validation engines.

6. Select the validation engine.

7. Select the **Automatic validation** option if you want to validate the current unit when the automatic validation feature is enabled in the Preferences.

8. Choose the schema to be used during validation (the schema detected after parsing the document or a custom one).

**Validation Actions in the User Interface**

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

• Select the **Validate** (Alt Shift V, V) action from the XML menu, from the **Validation** toolbar drop-down list, or from the **Validate** submenu when invoking the contextual menu in the **Navigator** view. An error list is presented in the message panel. Markup of current document is checked to conform with the specified DTD, XML Schema, or Relax NG schema rules. This action also re-parses the XML catalogs and resets the schema used for content completion.

• Select the **Validate (cached)** action from the XML menu or from the **Validation** toolbar drop-down list. This action caches the schema, allowing it to be reused for the next validation. Markup of the current document is checked to conform with the specified DTD, XML Schema, or Relax NG schema rules.

   **Note:** Automatic validation also caches the associated schema.

• Select the **Validate with...** action from the XML menu, from the **Validation** toolbar drop-down list, or from the **Validate** submenu when invoking the contextual menu in the **Navigator** view. You can use this action to validate the current document using a schema of your choice (XML Schema, DTD, Relax NG, NVDL, Schematron schema), other than the associated one. An error list is presented in the message panel. Markup of current document is checked to conform with the specified schema rules.

• Select **Validate with Schema...** from the **Validate** submenu when invoking contextual menu in the **Navigator** view to choose a schema and validate all selected files with it.
To open the schema used for validating the current document, select the **Open Associated Schema** action from the XML menu.

To clear the error markers added to the Problems view in the last validation, select **Clear Validation Markers** from the Validate submenu when invoking the contextual menu in the Navigator view.

**Tip:** If a large number of validation errors are detected and the validation process takes too long, you can limit the maximum number of reported errors in the Preferences page.

### Resolving References to Remote Schemas with an XML Catalog

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

```xml
<?xml-model href="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng" type="application/xml"
schematypens="http://relaxng.org/ns/structure/1.0"/>
```

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

```xml
<uri name="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng"
     uri="rng/docbook.rng"/>
```

An XML catalog can be used also to map a W3C XML Schema specified with an URN in the xsi:schemaLocation attribute of an XML document to a local copy of the schema. For example, if the XML document specifies the schema with:

```xml
<topic xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

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

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

### Document Navigation

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

#### Folding of the XML Elements

An XML document is organized as a tree of elements. When working on a large document you can collapse some elements leaving in the focus only the ones you need to edit. Expanding and collapsing works on individual elements: expanding an element leaves the child elements unchanged.
Figure 40: Folding of the XML Elements

To toggle the folded state of an element click on the special mark displayed in the left part of the document editor next to the start tag of that element or click on the action **Toggle fold (Ctrl Alt Y)** available from the contextual menu.

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

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

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

To watch our video demonstration about the folding support in Oxygen XML Developer plugin, go to [http://oxygenxml.com/demo/FoldingSupport.html](http://oxygenxml.com/demo/FoldingSupport.html).

### Outline View

The Outline view offers the following functionality:

- **XML Document Overview** on page 93
- **Outline Specific Actions** on page 93
- **Modification Follow-up** on page 94
- **Document Structure Change** on page 94
- **Document Tag Selection** on page 95
XML Document Overview

The Outline view displays a general tag overview of the current edited XML document. It also shows the correct hierarchical dependencies between the tag elements. This functionality makes it easier for the user to be aware of the document structure and the way tags are nested.

The Outline view allows you to:

- Insert or delete nodes using pop-up menu actions.
- Move elements by dragging them to a new position in the tree structure.
- Highlight elements in the Author editor area.

Note: The Outline view is synchronized with the Author editor area. When you make a selection in the Author editor area, the corresponding elements of the selection are highlighted in the Outline view and vice versa. This functionality is available both for single and multiple selection. To deselect one of the elements, use Ctrl Click (Command Click on OS X).

Document errors (such as an element inserted in an invalid position, or a wrong attribute name, or a missing required attribute value) are highlighted in the Outline tree:

- A red exclamation mark decorates the element icon.
- A dotted red underline decorates the element name and value.
- A tooltip provides more information about the nature of the error, when you hover with the mouse pointer over the faulted element.

Outline Specific Actions

The following actions are available in the:

Filter returns exact matches
- The text filter of the Outline view returns only exact matches.

Flat presentation mode of the filtered results
- When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions
- Show/hide comments and processing instructions in the Outline view.
Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from the Outline preferences panel.

Configure displayed attributes

Displays the XML Structured Outline preferences page.

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Modification Follow-up

When you edit a document, the Outline view dynamically follows the changes that you make, displaying the node that you modify in the middle of the view. This functionality gives you great insight on the location of your modifications in the document that you edit.

Document Structure Change

Entire XML elements can be moved or copied in the edited document using only the mouse in the Outline view in drag-and-drop operations. Several drag and drop actions are possible:

- If you drag an XML element in the Outline view and drop it on another one in the same panel then the dragged element will be moved after the drop target element.
- If you hold the mouse pointer over the drop target for a short time before the drop then the drop target element will be expanded first and the dragged element will be moved inside the drop target element after its opening tag.
- You can also drop an element before or after another element if you hold the mouse pointer towards the upper or lower part of the targeted element. A marker will indicate whether the drop will be performed before or after the target element.
- If you hold down the (Ctrl (Command on OS X)) key after dragging, there will be performed a copy operation instead of a move one.

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

The Contextual Menu of the Outline View

The following actions are available from the contextual menu in the Outline view:

Edit Attributes...

Allows you to edit all the attributes of a selected node.

The Append Child..., Insert Before..., and Insert After... actions allow you to quickly insert new tags into the document at the location of the currently selected element. When you select any of these three actions, a content completion window is invoked that offers a list of elements that can be inserted.

Append Child...

Invokes a content completion list with the names of all the elements that are allowed by the associated schema and inserts your selection as a child of the current element.

Insert Before...

Invokes a content completion list with the names of all the elements that are allowed by the associated schema and inserts your selection immediately before the current element, as a sibling.

Insert After...

Invokes a content completion list with the names of all the elements that are allowed by the associated schema and inserts your selection immediately after the current element, as a sibling.
The **Cut**, **Copy**, and **Delete** actions are the same actions as the **Edit** menu actions with the same name, for the currently selected elements.

**Toggle Comment**
Encloses the currently selected element in an XML comment, if the element is not already commented. If it is already commented, this action will remove the comment.

**Expand More**
Expands the structure tree of the currently selected element.

**Collapse All**
Collapses all of the structure tree of the currently selected node.

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

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

**Large Documents**

Let’s consider the case of documenting a large project. It is likely to be several people involved. The resulting document can be few megabytes in size. How to deal with this amount of data in such a way the work parallelism would not be affected?

Fortunately, XML provides two solutions for this: DTD entities and XInclude. It can be created a master document, with references to the other document parts, containing the document sections. The users can edit individually the sections, then apply an XSLT stylesheet over the master and obtain the result files, let say PDF or HTML.

**Including Document Parts with DTD Entities**

There are two conditions for including a part using DTD entities:

- The master document should declare the DTD to be used, while the external entities should declare the XML sections to be referenced.
- The document containing the section must not define again the DTD.

A master document looks like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "../xml/docbookx.dtd" [
  <!ENTITY testing SYSTEM "testing.xml" > ]
> 
<book>
  <chapter> ... </chapter>
</book>
```

The referenced document looks like this:

```xml
<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:

```xml
... &testing; ...
```
When splitting a large document and including the separate parts in the master file using external entities, only the master file will contain the Document Type Definition (the DTD) or other type of schema. The included sections can't define again the schema because the main document will not be valid. If you want to validate the parts separately you have to use XInclude for assembling the parts together with the master file.

Including Document Parts with XInclude

XInclude is a standard for assembling XML instances into another XML document through inclusion. It enables larger documents to be dynamically created from smaller XML documents without having to physically duplicate the content of the smaller files in the main file. XInclude is targeted as the replacement for External Entities. The advantage of using XInclude is that, unlike the entities method, each of the assembled documents is permitted to contain a Document Type Declaration (DOCTYPE). This means that each file is a valid XML instance and can be independently validated. It also means that the main document to which smaller instances are included can be validated without having to remove or comment out the DOCTYPE, as is the case with External Entities. This makes XInclude a more convenient and effective method for managing XML instances that need to be stand-alone documents and part of a much larger project.

The main application for XInclude is in the document-oriented content frameworks such as manuals and Web pages. Employing XInclude enables authors and content managers to manage content in a modular fashion that is akin to Object Oriented methods used in languages such as Java, C++ or C#.

The advantages of modular documentation include: reusable content units, smaller file units that are easier to be edited, better version control and distributed authoring.

<table>
<thead>
<tr>
<th>Include a chapter in an article using XInclude</th>
</tr>
</thead>
</table>

Create a chapter file and an article file in the samples folder of the Oxygen XML Developer plugin install folder.

Chapter file (introduction.xml) looks like this:

```xml
<?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
<?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 declaration defines an entity that references a file containing the information to add the `xi` namespace to certain elements defined by the DocBook DTD.
- The `href` attribute of the `xi:include` element specifies that the `introduction.xml` file will replace the `xi:include` element when the document is parsed.
If the `introduction.xml` file cannot be found, the parser will use the value of the `xi:fallback` element - a FIXME message.

If you want to include only a fragment of a file in the master file, the fragment must be contained in a tag having an `xml:id` attribute and you must use an XPointer expression pointing to the `xml:id` value. For example if the master file is:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="test.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<test>
  <xi:include href="a.xml" xpointer="a1"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
</test>
```

and the `a.xml` file is:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<test>
  <a xml:id="a1">test</a>
</test>
```

after resolving the XPointer reference the document is:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="test.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<test>
  <a xml:id="a1" xml:base="a.xml">test</a>
</test>
```

The XInclude support in Oxygen XML Developer plugin is enabled by default. To toggle it, open the Preferences dialog box and go to XML > XML Parser > Enable XInclude processing. When enabled, Oxygen XML Developer plugin will be able to validate and transform documents comprised of parts added using XInclude.

## Working with XML Catalogs

An XML Catalog maps a system ID or an URI reference pointing to a resource (stored either remotely or locally) to a local copy of the same resource. If XML processing relies on external resources (like referenced schemas and stylesheets, for example), the use of an XML Catalog becomes a necessity when Internet access is not available or the Internet connection is slow.

Oxygen XML Developer plugin supports any XML Catalog file that conforms to one of:

1. OASIS XML Catalogs Committee Specification v1.1
2. OASIS Technical Resolution 9401:1997 including the plain-text flavor described in that resolution

The version 1.1 of the OASIS XML Catalog specification introduces the possibility to map a system ID, a public ID or a URI to a local copy using only a suffix of the ID or URI used in the actual document. This is done using the catalog elements `systemSuffix` and `uriSuffix`.

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

### Table 2: Catalog Mappings

<table>
<thead>
<tr>
<th>Document</th>
<th>Referenced Resource</th>
<th>Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML DTD</td>
<td>system or public</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Prefer option controls which one of the mappings should be used.</td>
<td></td>
</tr>
<tr>
<td>XML Schema</td>
<td>The following strategy is used (if one step fails to provide a resource, the next is applied):</td>
<td></td>
</tr>
<tr>
<td>Relax NG</td>
<td>1. resolve the schema using URI catalog mappings.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. resolve the schema using system catalog mappings.</td>
<td></td>
</tr>
</tbody>
</table>
### Mappings Referenced

<table>
<thead>
<tr>
<th>Document Resource</th>
<th>Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schematron NVDL</td>
<td>This happens only if the <em>Resolve schema locations also through system mappings</em> option is enabled (it is by default).</td>
</tr>
<tr>
<td>XSL</td>
<td>resolve the root <em>namespace</em> using <em>URI</em> catalog mappings.</td>
</tr>
<tr>
<td>CSS</td>
<td><em>URI</em></td>
</tr>
<tr>
<td>XML Schema</td>
<td>The following strategy is used (if one step fails to provide a resource, the next is applied):</td>
</tr>
<tr>
<td>Relax NG</td>
<td>1. resolve schema reference using <em>URI</em> catalog mappings.</td>
</tr>
<tr>
<td></td>
<td>2. resolve schema reference using <em>system</em> catalog mappings.</td>
</tr>
<tr>
<td></td>
<td>This happens only if the <em>Resolve schema locations also through system mappings</em> option is enabled (it is by default).</td>
</tr>
<tr>
<td></td>
<td>3. resolve schema <em>namespace</em> using <em>uri</em> catalog mappings.</td>
</tr>
<tr>
<td></td>
<td>This happens only if the <em>Process namespaces through URI mappings for XML Schema</em> option is enabled (it is not by default).</td>
</tr>
</tbody>
</table>

An XML Catalog file can be created quickly in Oxygen XML Developer plugin starting from the two XML Catalog document templates called *OASIS XML Catalog 1.0* and *OASIS XML Catalog 1.1* and available in the document templates dialog.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog
    PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
    "http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <!-- Use "system" and "public" mappings when resolving DTDs -->
  <system
    systemId="http://www.docbook.org/xml/4.4/docbookx.dtd"
    uri="frameworks/docbook/4.4/dtd/docbookx.dtd"/>
  <!-- The "systemSuffix" matches any system ID ending in a specified string -->
  <systemSuffix
    systemIdSuffix="docbookx.dtd"
    uri="frameworks/docbook/4.4/dtd/docbookx.dtd"/>
  <!-- Use "uri" for resolving XML Schema and XSLT stylesheets -->
  <uri
    name="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng"
    uri="frameworks/docbook/5.0/rng/docbookrng.xsl"/>
  <!-- The "uriSuffix" matches any URI ending in a specified string -->
  <uriSuffix
    uriSuffix="docbook.xsl"
    uri="frameworks/docbook/xsl/fo/docbook.xsl"/>
</catalog>
```

Oxygen XML Developer plugin comes with a built-in catalog set as default, but you can also create your own one. Oxygen XML Developer plugin looks for a catalog in the following order:

- user-defined catalog set globally in the *XML Catalog preferences* page.
- user-defined catalog set at document type level, in the *Document Type Association preferences pages*.
- built-in catalogs.

An XML catalog can be used to map a W3C XML Schema specified with an URN in the `xsi:noNamespaceSchemaLocation` attribute of an XML document to a local copy of the schema.
Considering the following XML document code snippet:

```
  ...
</topic>
```

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

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

## Resolve Schemas Through XML Catalogs

Oxygen XML Developer plugin resolves the location of a schema in the following order:

- First, it attempts to resolve the schema location as a URI (uri, uriSuffix, rewriteURI from the XML catalog). If this succeeds, the process end here.
- If the Resolve schema locations also through system mappings option is selected, it attempts to resolve the schema location as a systemID (system, systemSuffix, rewriteSuffix, rewriteSystem from the XML catalog). If this succeeds, the process ends here.
- If the Process namespace through URI mappings for XML Schema option is selected, it attempts to resolve the schema location as a URI (uri, uriSuffix, rewriteURI from the XML catalog). If this succeeds, the process ends here.
- If none of these succeeds, the actual schema location is used.

## XML Resource Hierarchy/Dependencies View

The Resource Hierarchy / Dependencies view allows you to easily see the hierarchy / dependencies for an XML document. The tree structure presented in this view is built based on the XInclude and External Entity mechanisms. To define the scope for calculating the dependencies of a resource, click Configure dependencies search scope on the Resource Hierarchy/Dependencies toolbar.

To open this view, go to Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies. As an alternative, right click the current document and either select Resource Hierarchy or Resource Dependencies.

![Resource Hierarchy/Dependencies View](https://via.placeholder.com/150)

**Figure 42: Resource Hierarchy/Dependencies View - Hierarchy for Syncro phone v1.xml**
The build process for the dependencies view is started with the **Resource Dependencies** action available on the contextual menu.

![Resource Dependencies View](image)

**Figure 43: Resource Hierarchy/Dependencies View - Dependencies for Insert battery.xml**

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

- **Refresh**
  - Refreshes the Hierarchy/Dependencies structure.

- **Stop**
  - Stops the hierarchy/dependencies computing.

- **Show Hierarchy**
  - Allows you to choose a resource to compute the hierarchy structure.

- **Show Dependencies**
  - Allows you to choose a resource to compute the dependencies structure.

- **Configure**
  - Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

- **History**
  - Provides access to the list of previously computed dependencies. Use the ✪ Clear history button to remove all items from this list.

The contextual menu contains the following actions:

- **Open**
  - Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

- **Copy location**
  - Copies the location of the resource.

- **Move resource**
  - Moves the selected resource.

- **Rename resource**
  - Renames the selected resource.

- **Show Resource Hierarchy**
  - Shows the hierarchy for the selected resource.

- **Show Resource Dependencies**
  - Shows the dependencies for the selected resource.

- **Add to Master Files**
  - Adds the currently selected resource in the Master Files directory.
Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

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

Note: The Move resource or Rename resource actions give you the option to update the references to the resource. Only the references made through the XInclude and External Entity mechanisms are handled.

Moving/Renaming XML Resources

When you select the Rename action in the contextual menu of the Resource/Hierarchy Dependencies view, the Rename resource dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the Move action from the contextual menu of the Resource/Hierarchy Dependencies view, the Move resource dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

Converting Between Schema Languages

The Generate/Convert Schema dialog box allows you to convert a DTD or Relax NG (full or compact syntax) schema or a set of XML files to an equivalent XML Schema, DTD or Relax NG (full or compact syntax) schema. Where perfect equivalence is not possible due to limitations of the target language, Oxygen XML Developer plugin generates an approximation of the source schema. Oxygen XML Developer plugin uses Trang multi-format converter to perform the actual schema conversions.

To open the Generate/Convert Schema dialog box, select the Generate/Convert Schema... (Ctrl Shift \ (Command Shift \ on OS X)) action from the XML menu.

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 44: Convert a Schema to Other Schema Language

The language of the target schema is specified with one of the four options in the **Output** panel. Here you can also choose the encoding, the maximum line width and the number of spaces for one level of indentation.

The conversion can be further fine-tuned by specifying more advanced options available from the **Advanced options** button. For example if the input is a DTD and the output is an XML Schema the following options are available:

**Input panel:**

- **xmlns** - Specifies the default namespace, that is the namespace used for unqualified element names.
- **xmlns** - Each row specifies the prefix used for a namespace in the input schema.
- **colon-replacement** - Replaces colons in element names with the specified chars when constructing the names of definitions used to represent the element declarations and attribute list declarations in the DTD.
- **element-define** - Specifies how to construct the name of the definition representing an element declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition.
- **inline-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, the conversion engine uses a wildcard that allows any element
- **generate-start** - Specifies whether the conversion engine should generate a start element. DTD’s do not indicate what elements are allowed as document elements. The conversion engine assumes that all elements that are defined but never referenced are allowed as document elements.
- **annotation-prefix** - Default values are represented using an annotation attribute `prefix:defaultValue` where prefix is the specified value and is bound to `http://relaxng.org/ns/compatibility/annotations/1.0` as defined by the RELAX NG DTD Compatibility Committee Specification. By default, the conversion engine will use a for prefix unless that conflicts with a prefix used in the DTD.
Output panel:

- **disable-abstract-elements** - Disables the use of abstract elements and substitution groups in the generated XML Schema. This can also be controlled using an annotation attribute.
- **any-process-contents** - One of the values: strict, lax, skip. Specifies the value for the `processContents` attribute of any elements. The default is skip (corresponding to RELAX NG semantics) unless the input format is DTD, in which case the default is strict (corresponding to DTD semantics).
- **any-attribute-process-contents** - Specifies the value for the `processContents` attribute of `anyAttribute` elements. The default is skip (corresponding to RELAX NG semantics).

### Formatting and Indenting XML Documents

Oxygen XML Developer plugin creates XML documents using several different **edit modes**. In **text mode**, you as the author decide how the XML file is formatted and indented. In the other modes, and when you switch between modes, Oxygen XML Developer plugin must decide how to format and indent the XML. Oxygen XML Developer plugin will also format and indent your XML for you in text mode if you use one of the Format and Indent options:

- **Document > Source > Format and Indent** - formats and indents the whole document.
- **Document > Source > Indent Selection** - indents the current selection (but does not add line breaks)
- **Document > Source > Format and Indent Element** - formats and indents the current element (the inmost nested element which contains the current caret) and its child-elements.

A number of settings affect how Oxygen XML Developer plugin formats and indents XML. Many of these settings have to do with how whitespace is handled.

### Significant and insignificant whitespace in XML

XML documents are text files that describe complex documents. Some of the white space (spaces, tabs, line feeds, etc.) in the XML document belongs to the document it describes (such as the space between words in a paragraph) and some of it belongs to the XML document (such as a line break between two XML elements). Whitespace belonging to the XML file is called **insignificant whitespace**. The meaning of the XML would be the same if the insignificant whitespace were removed. Whitespace belonging to the document being described is called **significant whitespace**.

Knowing when whitespace is significant or insignificant is not always easy. For instance, a paragraph in an XML document might be laid out like this:

```
<p>
NO Freeman shall be taken or imprisoned, or be disseised of his Freehold, or Liberties, or free Customs, or be outlawed, or exiled, or any other wise destroyed; nor will We not pass upon him, nor condemn him, but by lawful judgment of his Peers, or by the Law of the land.
We will sell to no man, we will not deny or defer to any man either Justice or Right.
</p>
```

By default, XML considers a single whitespace between words to be significant, and all other whitespace to be insignificant. Thus the paragraph above could be written all on one line with no spaces between the start tag and the first word or between the last word and the end tag and the XML parser would see it as exactly the same paragraph. Removing the insignificant space in markup like this is called **normalizing space**.

But in some cases, all the spaces inside an element should be treated as significant. For example, in a code sample:

```
<codeblock>
class HelloWorld
  {
    public static void main(String args[])
    {
      System.out.println("Hello World");
    }
  }
</codeblock>
```

Here every whitespace character between the `codeblock` tags should be treated as significant.
How Oxygen XML Developer plugin determines when whitespace is significant

When Oxygen XML Developer plugin formats and indents an XML document, it introduces or removes insignificant whitespace to produce a layout with reasonable line lengths and elements indented to show their place in the hierarchy of the document. To correctly format and indent the XML source, Oxygen XML Developer plugin needs to know when to treat whitespace as significant and when to treat it as insignificant. However it is not always possible to tell this from the XML source file alone. To determine what whitespace is significant, Oxygen XML Developer plugin assigns each element in the document to one of four categories:

Ignore space
In the ignore space category, all whitespace is considered insignificant. This generally applies to content that consists only of elements nested inside other elements, with no text content.

Normalize space
In the normalize space category, a single whitespace character between character strings is considered significant and all other spaces are considered insignificant. This generally applies to elements that contain text content only. This content can be normalized by removing insignificant whitespace. Insensitive whitespace may then be added to format and indent the content.

Mixed content
In the mixed content category, a single whitespace between text characters is considered significant and all other spaces are considered insignificant. However,

• Whitespace between two child elements embedded in the text is normalized to a single space (rather than to zero spaces as would normally be the case for a text node with only whitespace characters, or the space between elements generally).

• The lack of whitespace between a child element embedded in the text and either adjacent text or another child element is considered significant. That is, no whitespace can be introduced here when formatting and indenting the file.

For example:

<p>The file is located in &lt;i&gt;HOME&lt;/i&gt;/&lt;i&gt;USER&lt;/i&gt;/hello. This is a &lt;strong&gt;big&lt;/strong&gt;&lt;emphasis&gt;deal&lt;/emphasis&gt;.
</p>

In this example, whitespace should not be introduced around the i tags as it would introduce extra significant whitespace into the document. The space between the end &lt;/strong&gt; tag and the beginning &lt;emphasis&gt; tag should be normalized to a single space, not zero spaces.

Preserve space
In the preserve space category, all whitespace in the element is regarded as significant. No changes are made to the spaces in elements in this category. Note, however, that child elements may be in a different category, and may be treated differently.

Attribute values are always in the preserve space category. The spaces between attributes in an element tag are always in the default space category.

Oxygen XML Developer plugin consults several pieces of information to assign an element to one of these categories. An element is always assigned to the most restrictive category (from Ignore to Preserve) that it is assigned to by any of the sources Oxygen XML Developer plugin consults. For instance, if the element is named on the Default elements list (as described below) but it has an xml:space="preserve" attribute in the source file, it will be assigned to the preserve space category. If an element has the xml:space="default" attribute in the source, but is listed on the Mixed content elements list, it will be assigned to the mixed content category.

To assign elements to these categories, Oxygen XML Developer plugin consults information from the following sources:
If the XML element contains the `xml:space` attribute, the element is promoted to the appropriate category based on the value of the attribute.

**Schema aware formatting**

If a schema is available for the XML document, Oxygen XML Developer plugin can use information from the schema to promote the element to the appropriate category. For example:

- If the schema declares an element to be of type `xs:string`, the element will be promoted to the preserve space category because the string built-in type has the whitespace facet with the value preserve.

- If the schema declares an element to be mixed content, it will be promoted to the mixed content category.

Schema aware formatting can be turned on and off.

- To turn it on or off for the Text editing mode, open the Preferences dialog box and go to Editor > Format > XML > Schema aware format and indent.

**Preserve space elements list**

If an element is listed in the Preserve space list in the XML formatting preferences, it is promoted to the preserve space category.

**Default space elements list**

If an element is listed in the Default space list in the XML formatting preferences, it is promoted to the default space category.

**Mixed content elements list**

If an element is listed in the Mixed content list in the XML formatting preferences, it is promoted to the mixed content category.

**Element content**

If an element contains mixed content, that is, a mix of text and other elements, it is promoted to the mixed content category. (Note that, in accordance with these rules, this happens even if the schema declares the element to have element only content.)

If an element contains text content, it is promoted to the default space category.

**Text node content**

If a text node contains any non-whitespace characters then the text node is promoted to the normalize space category.

**How Oxygen XML Developer plugin formats and indents XML**

You can control how Oxygen XML Developer plugin formats and indents XML documents. This can be particularly important if you store your XML document in a version control system, as it allows you to limit the number of trivial changes in spacing between versions of an XML document. The following settings pages control how XML documents are formatted:

- **Format Preferences** on page 506
- **XML Formatting Preferences** on page 507
- **Whitespaces Preferences** on page 508

**When Oxygen XML Developer plugin formats and indents XML**

Oxygen XML Developer plugin formats and indents a document, or part of it, on the following occasions:

- In text mode when you select one of the format and indent options (Document > Source > Format and Indent, Document > Source > Indent Selection, or Document > Source > Format and Indent Element).
- When saving documents in Design mode.
• When switching from Design mode to another mode.
• When saving or switching to Text mode from Grid mode, if the option Editor / Edit modes / Grid / Format and indent when passing from grid to text or on save is selected.

Setting an Indent Size to Zero

Oxygen XML Developer plugin will automatically format and indent documents at certain times. This includes indenting the content from the margin to reflect its structure. In some cases you may not want your content indented. To avoid your content being indented, you can set an indent size of zero.

Note: Changing the indent size does not override the rules that Oxygen XML Developer plugin uses for handling whitespace when formatting and indenting XML documents. Indents in elements that require whitespace to be maintained will not have their indent changed by these settings.

There are two cases to consider.

Maintaining zero indent in documents with zero indent

If you have existing documents with zero indent and you want Oxygen XML Developer plugin to maintain a zero indent when editing or formatting those documents:

1. Open the Preferences dialog box and go to Editor > Format.
2. Select Detect indent on open.
3. Select Use zero-indent if detected.

Oxygen XML Developer plugin will examine the indent of each document as it is opened and if the indent is zero for all lines, or for nearly all lines, a zero indent will be used when formatting and indenting the document. Otherwise, Oxygen XML Developer plugin will use the indent closest to what it detects in the document.

Enforcing zero indent for all documents

If you want all documents to be formatted with zero indent, regardless of their current indenting:

1. Open the Preferences dialog box and go to Editor > Format.
2. Deselect Detect indent on open.
3. Set Indent size to 0.

All documents will be formatted and indented with an indent of zero.

Warning: Setting the indent size to zero can change the meaning of some file types, such as Python source files.

Format and Indent (Pretty Print) Multiple Files

Oxygen XML Developer plugin provides support for formatting and indenting (Pretty Print) multiple files at once. This action is available for any document in XML format, as well as for XQuery, CSS, JavaScript, and JSON documents.

To format and indent multiple files, use the Format and Indent action that is available in the contextual menu of the Navigator view. This opens the Format and Indent dialog box that allows you to configure options for the action.
Figure 45: The Format and Indent Dialog Box

The Scope section allows you choose from the following scopes:

- **All opened files** - The pretty print is performed in all opened files.
- **Directory of the current file** - All the files in the folder of the current edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - Pretty prints the files located at a specified path.

The Options section includes the following options:

- **File filter** - Allow you to filter the files from the selected scope.
- **Recurse subdirectories** - When enabled, the pretty print is performed recursively for the specified scope. The one exception is that this option is ignored if the scope is set to All opened files.
- **Include hidden files** - When enabled, the pretty print is also performed in the hidden files.
- **Make backup files with extension** - When enabled, Oxygen XML Developer plugin makes backup files of the modified files. The default extension is .bak, but you can change the extension as you prefer.

Editing Modular XML Files in the Master Files Context

Smaller interrelated modules that define a complex XML modular structure cannot be correctly edited or validated individually, due to their interdependency with other modules. Oxygen XML Developer plugin provides the support for defining the main module (or modules), allowing you to edit any file from the hierarchy in the context of the master XML files.

You can set a main XML document either using the master files support from the Navigator view, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Developer plugin warns you if the current module is not part of the dependencies graph computed for the main XML document. In this case, it considers the current module as the main XML document.

The advantages of editing in the context of a master file include:

- correct validation of a module in the context of a larger XML structure;
- **Content Completion Assistant** displays all collected entities and IDs starting from the master files;
- Oxygen XML Developer plugin uses the schema defined in the master file when you edit a module which is included in the hierarchy through the **External Entity** mechanism;
• the master files defined for the current module determines the scope of the search and refactoring actions for ID/IDREFS values and for updating references when renaming/moving a resource. Oxygen XML Developer plugin performs the search and refactoring actions in the context that the master files determine, improving the speed of execution.

To watch our video demonstration about editing modular XML files in the master files context, go to http://oxygenxml.com/demo/Working_With_XML_Modules.html.

Managing ID/IDREFS.

Oxygen XML Developer plugin allows you to search for ID declarations and references (IDREFS) and to define the scope of the search and refactor operations. These operations are available for XML documents that have an associated DTD, XML Schema, or Relax NG schema.

Highlight IDs Occurrences in Text Mode

To see the occurrences of an ID in an XML document in the Text mode, place the cursor inside the ID declaration or reference. The occurrences are marked in the vertical side bar at the right of the editor. Click a marker on the side bar to navigate to the occurrence that it corresponds to. The occurrences are also highlighted in the editing area.

Note: Highlighted ID declarations are rendered with a different color than highlighted ID references. To customize these colors or disable this feature, open the Preferences dialog box and go to Editor > Mark Occurrences.

Search and Refactor Actions for ID/IDREFS

Oxygen XML Developer plugin offers full support for managing ID/IDREFS through the search and refactor actions available in the contextual menu. In Text mode, these actions are available in the Quick Assist menu as well.

The search and refactor actions from the contextual menu are grouped in the Manage IDs section:

Rename

Renames the ID and all its occurrences. Selecting this action opens the Rename XML ID dialog. This dialog lets you insert the new ID value and choose the scope of the rename operation. For a preview of the changes you are about to make, click Preview. This opens the Preview dialog, which presents a list with the files that contain changes and a preview zone of these changes.

Rename in File

Renames the ID you are editing and all its occurrences from the current file.

Note: Available in the Text mode only.

Search References in

Searches for the references of the ID. Selecting this action opens the Select the scope for the Search and Refactor operations.

Search References

Searches for the references of the ID. By default, the scope of this action is the current project. In case you configure a scope using the Select the scope for the Search and Refactor operations dialog, this scope will be used instead.

Search Declarations in

Searches for the declaration of the ID reference. Selecting this action opens the Select the scope for the Search and Refactor operations.

Search Declarations

Searches for the declaration of the ID reference. By default, the scope of this action is the current project. In case you configure a scope using the Select the scope for the Search and Refactor operations dialog, this scope will be used instead.

Search Occurrences in file

Searches for the declaration an references of the ID in the current document.
Note: A quick way to navigate to the declaration of an ID in Text mode is to move the cursor over an ID reference and use the Ctrl Click (Command Click on OS X) navigation.

Selecting an ID for which you execute search or refactor operations differs from the Text mode to the Author mode. In the Text mode you position the caret inside the declaration or reference of an ID. In the Author mode Oxygen XML Developer plugin collects all the IDs by analyzing each element from the path to the root. In case more IDs are available, you are prompted to choose one of them.

![Figure 46: Selecting an ID in the Author Mode](image)

### Quick Assist Support for ID/IDREFS in Text Mode

The Quick Assist support is activated automatically when you place the caret inside and ID or an IDREF. To access it, click the yellow bulb help marker placed on the caret line, in the line number stripe of the editor. You can also invoke the quick assist menu if you press Ctrl + 1 (Meta 1 on Mac OS X) on your keyboard.

The following actions are available:

- **Rename in**
  - Renames the ID and all its occurrences. Selecting this action opens the Rename XML ID dialog. This dialog lets you insert the new ID value and choose the scope of the rename operation. For a preview of the changes you are about to make, click Preview. This opens the Preview dialog, which presents a list with the files that contain changes and a preview zone of these changes.

- **Search Declarations**
  - Searches for the declaration of an ID reference. By default, the scope of this action is the current project. In case you configure a scope using the Select the scope for the Search and Refactor operations dialog, this scope will be used instead.

- **Search References**
  - Searches for the references of the ID. By default, the scope of this action is the current project. In case you configure a scope using the Select the scope for the Search and Refactor operations dialog, this scope will be used instead.

- **Change scope**
  - Opens the Select the scope for the Search and Refactor operations dialog;

- **Rename in File**
  - Renames the ID you are editing and all its occurrences from the current file.

  Note: Available in the Text mode only.

- **Search Occurrences**
  - Searches for the declaration of an ID located at the caret position in the current document.

### Search and Refactor Operations Scope

The scope is a collection of documents that define the context of a search and refactor operation. To control it you can use the Change scope operation, available in the Quick Fix action set or on the Resource Hierarchy/Dependency View toolbar. You can restrict the scope to the current project or to one or multiple working sets. The Use only Master
Files, if enabled check-box allows you to restrict the scope of the search and refactor operations to the resources from the Master Files directory. Click read more for details about the Master Files support.

Figure 47: Change Scope Dialog

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

Viewing Status Information

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

Figure 48: The Console view messages

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

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

XML Editor Specific Actions

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

Edit Actions

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

- contextual menu of current editor > Toggle comment Ctrl / (Command / on OS X) - Comments the current selection of the current editor. If the selection already contains a comment the action removes the comment from around the selection. If there is no selection in the current editor and the cursor is not positioned inside a comment the current line is commented. If the cursor is positioned inside a comment then the commented text is uncommented.
Select Actions

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

- **contextual menu of current editor > Select > Element** - Selects the entire current element;
- **contextual menu of current editor > Select > Content** - Selects the content of the current element, excluding the start tag and end tag. If it is applied repeatedly, starts with selecting the XML element from the cursor position and extends the selection to the ancestor XML elements. Each execution of the action extends the current selection to the surrounding element;
- **contextual menu of current editor > Select > Attributes** - Selects all the attributes of the current element;
- **contextual menu of current editor > Select > Parent** - Selects the parent element of the current element;
- Double click an element or processing instruction - If the double click is done before the start tag of an element or after the end tag of an element then all the element is selected by the double click action. If it is done after the start tag or before the end tag then only the element content without the start tag and end tag is selected;
- Double click an attribute in Text mode - If the double click is performed before the start tag of an attribute or after its end tag, the entire attribute is selected by the double click action. If it is performed after the start tag or before the end tag, only the attribute content (without the start tag and end tag) is selected;
- Double click after the opening quote or before the closing quote of an attribute value - Select the whole attribute value.

Source Actions

The following actions are available from the **Source** submenu when invoking the contextual menu in Text mode:

**Join and Normalize Lines**

For the current selection, this action joins the lines by replacing the *line separator* with a single space character. It also normalizes the whitespaces by replacing a sequence of such characters with a single space.

- **Shift Right Tab**
  Shifts the currently selected block to the right.

- **Shift Left Shift Tab**
  Shifts the currently selected block to the left.

- **Escape Selection ...**
  Escapes a range of characters by replacing them with the corresponding character entities.

- **Unescape Selection ...**
  Replaces the character entities with the corresponding characters.

- **Indent selection Ctrl I (Command I on OS X)**
  Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences*.

- **Format and Indent Element Ctrl Shift I**
  Pretty prints the element that surrounds the current caret position.

- **Insert XInclude**
  Displays a dialog that allows you to browse and select the content to be included and automatically generates the corresponding XInclude instruction.

  **Note:** In the **Author** mode, this dialog presents a preview of the inserted document as an author page in the **preview** tab and as a text page in the **Source** tab. In the **Text** mode, the **Source** tab is presented.
Import entities list
Displays a dialog that allows you to select a list of files as sources for external DTD entities. The internal subset of the DOCTYPE declaration of your document will be updated with the chosen entities. For instance, choosing the files chapter1.xml and chapter2.xml inserts the following section in the DOCTYPE:

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

Format and Indent
Performs a format and indent (pretty print) action on the current document.

Canonicalize
Opens the Canonicalize dialog that allows you to select a canonicalization algorithm to standardize the format of the document.

Sign
Opens the Sign dialog that allows you to configure a digital signature for the document.

Verify Signature
Allows you to specify the location of a file to verify its digital signature.

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

- **Show Definition**, available from the XML menu. Moves the cursor to the definition of the current element or attribute in the schema (DTD, XML Schema, Relax NG schema) associated with the edited XML document. In case the current attribute is "type" belonging to the "http://www.w3.org/2001/XMLSchema-instance" namespace, the cursor is moved in the XML schema, to the definition of the type referenced in the value of the attribute.

  Note: Alternatively you can **Ctrl Click (Command Click on OS X)** on an element or attribute name to invoke the Show Definition action.

- **Copy XPath (Ctrl Shift . (Command Shift . on OS X))**, available from the contextual menu of the current editor. Copies the XPath expression of the current element or attribute from the current editor to the clipboard.

- **Go to Matching Tag (Ctrl Shift G (Command Shift G on OS X))**, available from the Go to submenu when invoking the contextual menu of the current editor. Moves the cursor to the end tag that matches the start tag, or vice versa.

- **Go after Next Tag (Ctrl ] (Command ] on OS X))**, available from the Go to submenu when invoking the contextual menu of the current editor. Moves the cursor to the end of the next tag.

- **Go after Previous Tag (Ctrl [ (Command [ on OS X))**, available from the Go to submenu when invoking the contextual menu of the current editor. Moves the cursor to the end of the previous tag.

- **Associate XSLT/CSS Stylesheet...**, available from the XML menu. 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. The referenced stylesheet is used for rendering the document when opened in a Web browser. Referencing the XSLT file is also useful for automatic detection of the XSLT stylesheet when there is no scenario associated with the current document.

  When associating the CSS stylesheet, the user can also specify a title for it if it is an alternate one. Setting a **Title** for the CSS makes it the author's preferred stylesheet. Selecting the **Alternate** checkbox makes the CSS an alternate stylesheet.

  Oxygen XML Developer plugin fully implements the W3C recommendation regarding **Associating Style Sheets with XML documents**. See also **Specifying external style sheets** in HTML documents.

Also, you can use the **Ctrl Click (Command Click on OS X)** shortcut to open:

- Any absolute URLs (URLs that have a protocol) regardless of their location in the document.
• URI attributes such as: schemaLocation, noNamespaceSchemaLocation, href and others.
• Processing instructions used for associating resources, xml-models, xml-stylesheets.

Refactoring Actions

When editing an XML document, the following refactoring actions are available in the Refactoring submenu when invoking the contextual menu of the current editor:

• **Surround with Tags** (**Alt Shift E** ) - Allows you to choose a tag that encloses a selected portion of content. If there is no selection, the start and end tags are inserted at the caret position.
  - If the **Cursor position between tags** option is set, the caret is placed between the start and end tag.
  - If the **Cursor position between tags** option is not set, the caret is placed at the end of the start tag, in an insert-attribute position.

• **Surround with <tag>** (**Alt Shift /**) - Similar to the **Surround with Tags** action, except that it inserts the last tag used.

• **Rename Element** - The element from the caret position, and any elements with the same name, can be renamed according with the options from the Rename dialog box.

• **Rename Prefix** **Alt Shift P** - The prefix of the element from the caret position, and any elements with the same prefix, can be renamed according with the options from the Rename dialog box.
  - If you select the **Rename current element prefix** option, the application will recursively traverse the current element and all its children.
    - **Note:** For example, to change the xmlns:p1="ns1" association in the current element to xmlns:p5="ns1", if the xmlns:p1="ns1" association is applied on the parent element, then Oxygen XML Developer plugin will introduce xmlns:p5="ns1" as a new declaration in the current element and will change the prefix from p1 to p5. If p5 is already associated with another namespace in the current element, then the conflict will be displayed in a dialog box. By pressing OK, the prefix is modified from p1 to p5 without inserting a new declaration.
  - If you select the **Rename current prefix in all document** option, the application will apply the change on the entire document.
  - To also apply the action inside attribute values, check the **Rename also attribute values that start with the same prefix** checkbox.

• **Split element** - Split the element from the caret position into two identical elements. The caret must be inside the element.

• **Join elements** (**Alt Shift F**) - Joins the left and right elements relative to the current caret position. The elements must have the same name, attributes, and attributes values.

• **Delete element tags** (**Alt Shift ,**) - Deletes the start and end tag of the current element.

Smart Editing

The following helper actions are available in the XML editor:

• **Closing tag auto-expansion** - If you want to insert content into an auto closing tag like <tag/> deleting the / character saves some keystrokes by inserting a separate closing tag automatically and placing the cursor between the start and end tags: <tag></tag>

• **Auto-rename matching tag** - When you edit the name of the start tag, Oxygen XML Developer plugin will mirror-edit the name of the matching end tag. This feature can be controlled from the **Content Completion option page**.

• **Auto-breaking the edited line** - The **Hard line wrap option** breaks the edited line automatically when its length exceeds the maximum line length defined for the format and indent operation.

• **Indent on Enter** - The **Indent on Enter option** indents the new line inserted when Enter is pressed.
• **Smart Enter** - The **Smart Enter option** inserts an empty line between the start and end tags. If Enter is pressed between a start and an end tag the action places the cursor in an indented position on the empty line between the lines that contain the start and end tag.

• **Double click** - A double click selects a different region of text of the current document depending on the position of the click in the document:
  - If the click position is inside a start tag or an end tag then the entire element enclosed by that tag is selected.
  - If the click position is immediately after a start tag or immediately before an end tag then the entire content of the element enclosed by that tag is selected, including all the child elements but excluding the start tag and the end tag of the element.
  - Otherwise, the double click selects the entire current line of text.

**Syntax Highlight Depending on Namespace Prefix**

The **syntax highlight scheme of an XML file type** allows the configuration of a color per each type of token which can appear in an XML file. Distinguishing between the XML tag tokens based on the namespace prefix brings additional visual help in editing some XML file types. For example in XSLT stylesheets elements from different namespaces like XSLT, XHTML, XSL:FO or XForms are inserted in the same document and the editor panel can become cluttered. **Marking tags with different colors based on the namespace prefix** allows easier identification of the tags.

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

**Figure 49: Example of Coloring XML Tags by Prefix**

**Editor Highlights**

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

By default, Oxygen XML Developer plugin uses a different color for each type of highlight: **XPath, Find, Search References**, and **Search Declaratins**. You can customize these colors and the maximum number of highlights displayed in a document on the **Editor preferences page**. The default maximum number of highlights is 10000.

You are able to navigate in the current document through the highlights using one of the following methods:

• Clicking the markers from the range ruler, located at the right side of the document.

• Clicking the ✈ Next and Previous buttons from the bottom of the range ruler.

  ✉ **Note:** When there are multiple types of highlights in the document, the ✈ Next and Previous buttons navigate through highlights of the same type.

• Clicking the messages displayed in the **Results view**.

To remove the highlights, you can:

• Click the ✖ Remove all button from bottom of the range ruler.

• Close the results tab that contains the output of the action that generated the highlights.

• Click the ✖ Remove all button from the results panel.

  ✉ **Note:** Use the ✉ Highlight all results in editor button to either display all the highlights or hide them.
Batch Editing Actions on Highlights

Working with XML documents implies frequent changes to structure and content. You are often faced with a situation where you need to make a slight change in hundreds of places in the same document. Oxygen XML Developer plugin introduced a new feature, Manage Highlighted Content, designed to help you achieve that.

When you are in Text mode and you perform a search operation or apply an XPath that highlights more than one result, you can select the Manage Highlighted Content action from the contextual menu of any highlight in the document. In the sub-menu, the following options are available:

- **Modify All** - Use this option to modify in-place all the occurrences of the selected content. When you use this option, a thin rectangle replaces the highlights and lets you start editing;
  
  **Note:** In case you select a very large number of highlights that you want to modify using this feature, when you select this option, a dialog box informs you that you may experience performance issues. You have the option to either use the Find/Replace dialog box, or continue the operation.

- **Surround All** - Use this option to surround the content with a specific tag. This option opens the Tag dialog box. The Specify the tag drop-down presents all the available elements that you can choose from.

- **Remove All** - Removes all the highlighted content.

In case you right click a different part of the document than a highlight, you only have the option to select Modify All Matches.

XML Quick Fixes

The Oxygen XML Developer plugin Quick Fix support helps you resolve errors that appear in an XML document by offering quick fixes to problems such as missing required attributes or invalid elements. This section explains the quick fix support for XSD, Relax NG, and Schematron validation errors.

To activate this feature, place the caret in the highlighted area of text where a validation error occurs. If a Quick Fix is available for that particular error, the icon is displayed in the stripe on the left side of the editor. If you click this icon, Oxygen XML Developer plugin displays the list of available fixes. You can also invoke the quick fix menu by pressing Ctrl 1 (Command 1 on OS X) on your keyboard.

Whenever you make a modification in the XML document or you apply a fix, the list of quick fixes is recomputed to ensure that you always have valid proposals.

**Note:** A quick fix that adds an element inserts it along with required and optional elements, and required and fixed attributes, depending on how the Content Completion Assistant options are set.

Quick Fixes for XSD and Relax NG Errors

Oxygen XML Developer plugin offers quick fixes for common errors that appear in XML documents. Quick fixes are available for XML documents that are validated against XSD or Relax NG schemas.

**Note:** For XML documents validated against XSD schemas, the quick fixes are only available if you use the default Xerces validation engine.

Quick fixes are available in Text mode:

Oxygen XML Developer plugin provides quick fixes for numerous problems, including:

<table>
<thead>
<tr>
<th>Problem type</th>
<th>Available quick fixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A specific element is required in the current context</td>
<td>Insert the required element</td>
</tr>
<tr>
<td>Problem type</td>
<td>Available quick fixes</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>An element is invalid in the current context</td>
<td>Remove the invalid element</td>
</tr>
<tr>
<td>The content of the element should be empty</td>
<td>Remove the element content</td>
</tr>
<tr>
<td>An element is not allowed to have child elements</td>
<td>Remove all child elements</td>
</tr>
<tr>
<td>Text is not allowed in the current element</td>
<td>Remove the text content</td>
</tr>
<tr>
<td>A required attribute is missing</td>
<td>Insert the required attribute</td>
</tr>
<tr>
<td>An attribute is not allowed to be set for the current element</td>
<td>Remove the attribute</td>
</tr>
<tr>
<td>The attribute value is invalid</td>
<td>Propose the correct attribute values</td>
</tr>
<tr>
<td>ID value is already defined</td>
<td>Generate a unique ID value</td>
</tr>
<tr>
<td>References to an invalid ID</td>
<td>Change the reference to an already defined ID</td>
</tr>
</tbody>
</table>

**Schematron Quick Fixes (SQF)**

Oxygen XML Developer plugin provides support for Schematron Quick Fixes (SQF). They help you resolve errors that appear in XML documents that are validated against Schematron schemas by offering you solution proposals. The Schematron Quick Fixes are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. Specifically, they are associated with assert or report messages.

**Displaying the Schematron Quick Fix Proposals**

The defined Schematron Quick Fixes are displayed on validation errors in Text mode.

![Figure 50: Example of a Schematron Quick Fix](image)

**Refactoring XML Documents**

In the life cycle of XML documents there are instances when the XML structure needs to be changed to accommodate various needs. For example, when an associated schema is updated, an attribute may have been removed, or a new element added to the structure.

These types of situations cannot be resolved with a traditional Find/Replace tool, even if the tool accepts regular expressions. The problem becomes even more complicated if an XML document is computed or referenced from multiple modules, since multiple resources need to be changed.

To assist you with these types of refactoring tasks, Oxygen XML Developer plugin includes a specialized XML Refactoring tool that helps you manage the structure of your XML documents.

**XML Refactoring Tool**

The XML Refactoring tool is presented in the form of an easy to use wizard that is designed to reduce the time and effort required to perform various structure management tasks. For example, you can insert, delete, or rename an attribute in all instances of a particular element that is found in all documents within your project.

To access the tool, select the XML Refactoring... action from one of the following locations:

- The XML Tools menu.
- The Refactoring submenu from the contextual menu in the Navigator view.
The tool includes the following wizard pages:

**Refactoring operations**

The first wizard page displays, and allows you to select, the available operations, which are grouped by category. To search for an operation, you can use the filter text box at the top of the page.

**Configure Operation Parameters**

The next wizard page allows you to specify the parameters for the refactoring operation. The parameters are specific to the type of refactoring operation that is being performed. For example, to delete an attribute you need to specify the parent element and the qualified name of the attribute to be removed.
Scope and Filters

The last wizard page allows you to select the set of files that represent the input of the operation. You can select from predefined resource sets (such as the current file, your whole project, the current DITA Map hierarchy, etc.) or you can define your own set of resources by creating a working set.

The Filters section includes the following options:

- **Include files** - Allows you to filter the selected resources by using a file pattern. For example, to restrict the operation to only analyze build files you could use `build*.xml` for the file pattern.
- **Restrict only to known XML file types** - When enabled, only resources with a known XML file type will be affected by the operation.

If an operation takes longer than expected you can use the Stop button in the progress bar to cancel the operation.

**Note:** It is recommended that you use the Preview button to review all the changes that will be made by the refactoring operation before applying the changes.

**Warning:** After clicking the Finish button, the operation will be processed and Oxygen XML Developer plugin provides no automatic means for reverting the operations. Any Undo action will only revert changes on the current document.

Predefined Refactoring Operations

The XML Refactoring tool includes a variety of predefined operations that can be used for common refactoring tasks. They are grouped by category in the Refactoring operations wizard page and include the following operations:

Refactoring Operations for Attributes

1. **Add/Change attribute** - Use this operation to change the value of an attribute or insert a new one. To perform this operation, specify the following parameters:
   - The Local name and Namespace for the Parent element.
   - The Local name, Namespace, and Value of the affected Attribute.
   - One of the following choices for the Operation mode in the Options section:
     - Add the attribute in the parent elements where it is missing
     - Change the value in the parent elements where the attribute already exists
2. **Delete attribute** - Use this operation to remove one or more attributes. To perform this operation, specify the following parameters:
   - The **Local name** and **Namespace** for the **Parent element**.
   - The **Local name** and **Namespace** for the **Attribute** to be removed.

3. **Rename attribute** - Use this operation to rename an attribute. Specify the following parameters in the **Rename attribute** dialog box:
   - The **Local name** and **Namespace** for the **Parent element**.
   - The **Local name**, **Namespace**, and **New local name** of the **Attribute**.

4. **Replace in attribute value** - Use this operation to search for a text fragment inside an attribute value and change the fragment to a new value. To perform this operation, specify the following parameters:
   - The **Local name** and **Namespace** for the **Parent element**.
   - The **Local name** and **Namespace** for the **Attribute** to be modified.
   - The text **Fragments to Find** and **Replace with**.

   **Note:** You can use Perl-like regular expressions when specifying the text to find. The **Replace with** parameter can bind regular expression capturing groups ($1, $2, etc.) from the find pattern.

---

**Refactoring Operations for Elements**

1. **Delete element** - Use this operation to delete elements. To perform this operation, specify the following parameters:
   - The **Target elements** in the form of an XPath expression.

2. **Delete element content** - Use this operation to delete content of elements. To perform this operation, specify the following parameters:
   - The **Target elements** in the form of an XPath expression.

3. **Insert element** - Use this operation to insert new element. To perform this operation, specify the following parameters:
   - The **Local name** and **Namespace** for the **element** to be inserted.
   - The **Location** of the new element in the form of an XPath expression and its **Position**. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes. The possible selections in the **Position** drop-down list include **After**, **Before**, **First child**, or **Last child**.

4. **Rename element** - Use this operation to rename elements. To perform this operation, specify the following parameters:
   - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.
   - The **New local name** of the element.

5. **Unwrap element** - Use this operation to delete the surrounding tags of elements, while keeping their content unchanged. To perform this operation, specify the following parameters:
   - The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

6. **Wrap element** - Use this operation to surround elements with element tags. To perform this operation, specify the following parameters:
• The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the XML / XSLT-FO-XQuery / XPath preferences page where you can configure XPath options and declare namespace prefixes.

• The **Local name** and **Namespace** of the **Wrapper element**.

7. **Wrap element content** - Use this operation to surround the content of elements with element tags. To perform this operation, specify the following parameters:

• The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the XML / XSLT-FO-XQuery / XPath preferences page where you can configure XPath options and declare namespace prefixes.

• The **Local name** and **Namespace** of the **Wrapper element** in which its content will be wrapped.

## Refactoring Operations for XML Fragments

1. **Insert XML fragment** - Use this operation to insert an XML fragment. To perform this operation, specify the following parameters:

   • The **XML Fragment** to be inserted.

   • The **Location** of the new fragment in the form of an XPath expression and its **Position**. Use the link provided in the lower part of the wizard to open the XML / XSLT-FO-XQuery / XPath preferences page where you can configure XPath options and declare namespace prefixes. The possible selections in the **Position** drop-down list include **After**, **Before**, **First child**, or **Last child**.

2. **Replace element content with XML fragment** - Use this operation to replace the content of elements with an XML fragment. To perform this operation, specify the following parameters:

   • The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the XML / XSLT-FO-XQuery / XPath preferences page where you can configure XPath options and declare namespace prefixes.

   • The **XML Fragment** with which to replace the content of the target elements.

3. **Replace element with XML fragment** - Use this operation to replace elements with an XML fragment. To perform this operation, specify the following parameters:

   • The **Target elements** in the form of an XPath expression. Use the link provided in the lower part of the wizard to open the XML / XSLT-FO-XQuery / XPath preferences page where you can configure XPath options and declare namespace prefixes.

   • The **XML Fragment** with which to replace the target elements.

## Additional Notes

**Note:** There are some operations that allows `<ANY>` for the **local name** and **namespace** parameters. This value can be used to select an element or attribute regardless of its local name or namespace. Also, the `<NO_NAMESPACE>` value can be used to select nodes that do not belong to a namespace.

**Note:** Some operations have parameters that accept XPath expressions to match elements or attributes. In these XPath expressions you can only use the prefixes declared in the **Options > Preferences > XML > XSLT-FO-XQUERY > XPath** page. This preferences page can be easily opened by clicking on the link in the note (Each prefix used in an XPath expression must be declared in the Default prefix-namespace mappings section) at the bottom of the **Configure Operation Parameters** wizard page.

## Custom Refactoring Operations

If none of the predefined operations will help you accomplish a particular refactoring task, you can create a custom operation that is specific to your needs. For example, if you want to convert an attribute to an element and insert the element as the first child of the parent element, a custom refactoring operation needs to be created.
An XML Refactoring operation is defined as a pair of resources:

- An XQuery Update script file that Oxygen XML Developer plugin will run in order to refactor the XML files.
- An XML Operation Descriptor file that contains information about the operation, such as the name, description, and parameters.

![Diagram of an XML Refactoring Operation](image)

**Figure 51: Diagram of an XML Refactoring Operation**

All the defined operations are loaded by the XML Refactoring tool and presented in the Refactoring operations wizard page.

After the user chooses an operation and specifies its parameters, an XQuery Update transformation is processed over the input file. This transformation is executed in a safe mode, which implies the following:

- When loading the document:
  - The XInclude mechanism is disabled. This means that the resources included by using XInclude will not be visible in the transformation.
  - The DTD entities will be processed without being expanded.
  - The associated DTD will be not loaded, so the default attributes declared in the DTD will not be visible in the transformation.

- When saving the updated XML document:
  - The DOCTYPE will be preserved.
  - The DTD entities will be preserved as they are in the original document when the document is saved.
  - The attribute values will be kept in their original form without being normalized.
  - The spaces between attributes are preserved. Basically, the spaces are lost by an XML serialization since they are not considered important.

The result of this transformation overrides the initial input file.

**Note:** To achieve some of the previous goals, the XML Refactoring mechanism adds several attributes that are interpreted internally. The attributes belong to the http://oxygenxml.com/app/xml_refactory/additional_attributes namespace. These
attributes should not be taken into account when processing the input XML document, since they are discarded when the transformed document is serialized.

**Restriction:** Comments or processing instructions that are in any node before or after the root element cannot be modified by an XML Refactoring operation. In other words, XML Refactoring operations can only be performed on comments or processing instructions that are inside the root element.

Creating a Custom Refactoring Operation

To create a custom refactoring operation, follow these steps:

1. **Create an XQuery Update script file.**
2. **Create an XML Refactoring Operation Descriptor file.**
3. Store both files in one of the locations that Oxygen XML Developer plugin scans when loading the custom operations.

Once you run the XML Refactoring tool again, the custom operation appears in the Refactoring operations wizard page.

Custom Refactoring XQuery Update Script

The first step in creating a custom refactoring operation is to create an XQuery Update script. The easiest way to create this file is to use the New document wizard and create a new XQuery file.

There are cases when it is necessary to add parameters in the XQuery script. For instance, if you want to rename an element you may want to declare an external parameter associated with the name of the element to rename. To allow you to specify the value for these parameters, they need to be declared in the refactoring operation descriptor file that is associated with this operation.

**Note:** The XQuery Update processing is disabled by default in Oxygen XML Developer plugin. Thus, if you want to create or edit an XQuery Update script you should disable this facility by creating an XQuery transformation scenario and enable the Enable XQuery update option in the Saxon processor advanced options.

The next step in creating a custom refactoring operation is to create a custom operation descriptor file.

Custom Refactoring Operation Descriptor File

The second step in creating a custom refactoring operation is to create an operation descriptor file. The easiest way to do this is to use the New document wizard and choose the XML Refactoring Operation Descriptor template.

**Introduction to the Descriptor File**

This file contains information (such as name, description, and id) that is necessarily when loading an XML Refactoring operation. It also contains the path to the XQuery Update script that is associated with the particular operation through the script element.

You can specify a category for your custom operations to logically group certain operations. The category element is optional and if it is not included in the descriptor file, the default name of the category for the custom operations is Other operations.

The descriptor file is edited and validated against the following schema:

frameworks/xml_refactoring/operation_descriptor.xsd.

**Declaring Parameters in the Descriptor File**

If the XQuery Update script includes parameters, they should be declared in the parameters section of the descriptor file. All the parameters specified in this section of the descriptor file will be displayed in the XML Refactoring tool within the Configure Operation Parameters wizard page for that particular operation.

The value of the first description element in the parameters section will be displayed at the top of the Configure Operation Parameters wizard page.
To declare a parameter, specify the following information:

- **label** - This value is displayed in the user interface for the parameter.
- **name** - The parameter name used in the XQuery Update script and it should be the same as the one declared in the XQuery script.
- **type** - Defines the type of the parameter and how it will be rendered. There are several types available:
  - TEXT - Generic type used to specify a simple text fragment.
  - XPATH - Type of parameter whose value is an XPATH expression. For this type of parameter, Oxygen XML Developer plugin will use a text input with corresponding content completion and syntax highlighting.
    
    **Note:** The value of this parameter is transferred as plain text to the XQuery Update transformation without being evaluated. To evaluate it in the XQuery Update script you could use the `saxon:evaluate` Saxon extension function.
  
  **Note:** A relative XPath expression is converted to an absolute XPath expression by adding `//` before it (`//XPathExp`). This conversion is done before transferring the XPath expression to the XQuery Update engine.
  
  **Note:** When writing XPath expressions, you can only use prefixes declared in the Options > Preferences > XML > XSLT-FO-XQUERY > XPath options page.
  - NAMESPACE - Used for editing namespace values.
  - REG_EXP_FIND - Used when you want to match a certain text by using Perl-like regular expressions.
  - REG_EXP_REPLACE - Used along with REG_EXP_FIND to specify the replacement string.
  - XML_FRAGMENT - This type is used when you want to specify an XML fragment. For this type, Oxygen XML Developer plugin will display a text area specialized for inserting XML documents.
  - NC_NAME - The parameter for NC_NAME values. It is useful when you want to specify the local part of a QName for an element or attribute.
  - BOOLEAN - Used to edit boolean parameters.
  - TEXT_CHOICE - It is useful for parameters whose value should be from a list of possible values. Oxygen XML Developer plugin renders each possible value as a radio button.
  - **description** - The description of the parameter. It is used by the application to display a tooltip when you hover over the parameter.
  - **possibleValues** - Contains the list with possible values for the parameter and you can specify the default value, as in the following example:

    ```
    <possibleValues onlyPossibleValuesAllowed="true">
      <value name="before">Before</value>
      <value name="after" default="true">After</value>
      <value name="firstChild">First child</value>
      <value name="lastChild">Last child</value>
    </possibleValues>
    ```

**Specialized Parameters (elementParameter and attributeParameter)**

If you want to match elements or attributes, use the specialized parameters `elementParameter` or `attributeParameter`. If you use these parameters, Oxygen XML Developer plugin will propose all declared elements or attributes based on the schema associated with the currently edited file.

Example of an `elementParameter`:

```
<elementParameter id="elemID">
  <localName label="Name" name="element_localName" allowsAny="true">
    <description>The local name of the attribute's parent element.</description>
  </localName>
  <namespace label="Namespace" name="element_namespace" allowsAny="true">
    <description>The local name of the attribute's parent element</description>
  </namespace>
</elementParameter>
```
This parameter is used to specify elements by local name and namespace. For this type of parameter, the application displays two combo boxes with elements and namespaces collected from the associated schema of the currently edited file. The text from the `label` attribute is displayed in the application as label of the associated combo. The `name` attribute is used to specify the name of the parameter from the XQuery Update script. If you specify the `allowsAny` attribute, the application will propose `<ANY>` as a possible value for the `Name` and `Namespace` combo boxes.

Example of an `attributeParameter`:

```xml
<attributeParameter dependsOn="elemID">
  <localName label="Name" name="attribute_localName">
    <description>The name of the attribute to be converted.</description>
  </localName>
  <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
    <description>The namespace of the attribute to be converted.</description>
  </namespace>
</attributeParameter>
```

This parameter is used to specify attributes by local name and namespace. For this type of parameter, the application displays two combo boxes with attributes and their namespaces collected from the associated schema of the currently edited file. The text from the `label` attribute is displayed in the application as the label of the associated combo box.

Note: An `attributeParameter` is dependant upon an `elementParameter`. The list of attributes and namespaces are computed based on the selection in the `elementParameter` combo boxes.

Note: All predefined operations are loaded from the `installDir/refactoring` folder.

**Example of an XML Refactoring Operation**

To demonstrate creating a custom operation, consider that we have a task where we need to convert an attribute into an element and insert it inside another element. A specific example would be if you have a project with a variety of `image` elements where a deprecated `alt` attribute was used for the description and you want to convert all instances of that attribute into an element with the same name and insert it as the first child of the `image` element.

Thus, our task is to convert this attribute into an element with the same name and insert it as the first child of the `image` element.

```
<image>
  <alt>Insert the battery into the battery compartment.</alt>
</image>
```

A new operation requires an XQuery Update script and an XML Refactoring operation descriptor file.

**Sample XQuery Update Script for Creating a Custom Operation to Convert an Attribute to an Element**

The XQuery Update script does the following:

- Iterates over all elements from the document that have the specified local name and namespace.
• Finds the attribute that will be converted to an element.
• Computes the QName of the new element to be inserted and inserts it as the first child of the parent element.

```
(: XQuery document used to implement 'Convert attribute to element' operation from XML Refactoring tool. :)

declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "xml";
declare option output:indent "no";

(: Local name of the attribute's parent element. :) declare variable $element_localName as xs:string external;

(: Namespace of the attribute's parent element. :) declare variable $element_namespace as xs:string external;

(: The local name of the attribute to be converted :) declare variable $attribute_localName as xs:string external;

(: The namespace of the attribute to be converted :) declare variable $attribute_namespace as xs:string external;

(: Local name of the new element :) declare variable $new_element_localName as xs:string external;

(: Namespace of the new element :) declare variable $new_element_namespace as xs:string external;

(: Convert attribute to element :) for $node in //*

(: Find the attribute to convert :) let $attribute := $node/@*[local-name() = $attribute_localName and ($attribute_namespace = '<ANY>' or $attribute_namespace = namespace-uri())]

(: Compute the prefix for the new element to insert :) let $prefix :=

for $p in in-scope-prefixes($node)

where $new_element_namespace = namespace-uri-$for-prefix($p, $node)
return $p

(: Compute the QName for the new element to insert :) let $new_element_qName :=

if (empty($prefix) or $prefix[1] = '') then $new_element_localName
else $prefix[1] || ':' || $new_element_localName

where ('<ANY>' = $element_localName or local-name($node) = $element_localName) and
($element_namespace = '<ANY>' or $element_namespace = namespace-uri($node))

return

if (exists($attribute)) then

(insert node element QName($new_element_namespace, $new_element_qName))

(string($attribute)) as first into $node,

delete node $attribute)
else ()
```

Sample Operation Descriptor File for Creating a Custom Operation to Convert an Attribute to an Element

After you have developed the XQuery script, you have to create an XML Refactoring operation descriptor. This descriptor is used by application to load the operation details such as name, description, or parameters.

```
<?xml version="1.0" encoding="UTF-8"?>
<refactoringOperationDescriptor
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://oxygenxml.com/ns/xmlRefactoring"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="convert-attribute-to-element"
name="Convert attribute to element">
<description>Converts the specified attribute to an element. The new element will be inserted as first child of the attribute's parent element.</description>
<script type="XQUERY_UPDATE" href="convert-attribute-to-element.xq"/>
<parameters>
<description>Specify the attribute to be converted to element.</description>
<section label="Parent element">
<elementParameter id="elemID">
<localName label="Name" name="element_localName" allowsAny="true">
<description>The local name of the attribute's parent element.</description>
</localName>
<namespace label="Namespace" name="element_namespace" allowsAny="true">
<description>The local name of the attribute's parent element</description>
</namespace>
```
After you have created these files, copy them into a folder scanned by Oxygen XML Developer plugin when it loads the custom operation. When the XML Refactoring tool is started again, you will see the created operation.

Since various parameters can be specified, this custom operation can also be used for other similar tasks. The following image shows the parameters that can be specified in our example of the custom operation to convert an attribute to an element:

---

**Storing and Sharing Refactoring Operations**

Oxygen XML Developer plugin scans the following locations when looking for XML Refactoring operations to provide flexibility:

- A refactoring folder, created inside a directory that is associated to a framework you are customizing.
- Any folder. In this case, you need to open the Preferences dialog box, go to XML > XML Refactoring, and specify the same folder in the Load additional refactoring operations from text box.
- The refactoring folder from the Oxygen XML Developer plugin installation directory (`{oxygenInstallation Directory}/refactoring`).

**Sharing Custom Refactoring Operations**

The purpose of Oxygen XML Developer plugin scanning multiple locations for the XML Refactoring operations is to provide more flexibility for developers who want to share the refactoring operations with the other team members. Depending on your particular use case, you can attach the custom refactoring operations to other resources, such as frameworks or projects.
After storing custom operations, you can share them with other users by sharing the resources.

**Localizing XML Refactoring Operations**

Oxygen XML Developer plugin includes localization support for the XML refactoring operations.

The translation keys for the built-in refactoring operations are located in `[oXygen Installation Directory]/refactoring/i18n/translation.xml`.

The localization support is also available for custom refactoring operations. The following information can be translated:

- The operation name, description, and category.
- The description of the parameters element.
- The label, description, and possibleValues for each parameter.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Developer plugin scans the following locations to find the `translation.xml` files that are used to load the translation keys:

- A `refactoring/i18n` folder, created inside a directory that is associated to a customized framework.
- A `i18n` folder, created inside a directory that is associated to a customized framework.
- An `i18n` folder inside any specified folder. In this case, you need to open the Preferences dialog box, go to XML > XML Refactoring, and specify the folder in the Load additional refactoring operations from text box.
- The `refactoring/i18n` folder from the Oxygen XML Developer plugin installation directory (`[oXygen Installation Directory]/refactoring/i18n`).

---

**Example of a Refactoring Operation Descriptor File with i18n Support**

```
<?xml version="1.0" encoding="UTF-8"?>
<refactoringOperationDescriptor
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://oxygenxml.com/app/xml_refactory
 http://oxygenxml.com/app/xml_refactory/operation_descriptor.xsd"
 xmlns="http://oxygenxml.com/app/xml_refactory"
 id="remove_text_content"
 name="${i18n(Remove_text_content)}">
     <description>${i18n(Remove_text_content_description)}</description>
     <script type="XQUERY_UPDATE" href="remove_text_content.xq"/>
     <parameters>
         <parameter label="${i18n(Element_name)}" name="element_localName" type="NC_NAME">
             <description>${i18n(Element_name_descriptor)}</description>
             <possibleValues>
                 <value default="true" name="value1">${i18n(value_1)}</value>
                 <value name="value2">${i18n(value_2)}</value>
             </possibleValues>
         </parameter>
     </parameters>
 </refactoringOperationDescriptor>
```

---

**Editing XSLT Stylesheets**

This section explains the features of the XSLT editor.

To watch our video demonstration about basic XSLT editing and transformation scenarios in Oxygen XML Developer plugin, go to [http://oxygenxml.com/demo/XSL_Editing.html](http://oxygenxml.com/demo/XSL_Editing.html).

**Validating XSLT Stylesheets**

Oxygen XML Developer plugin performs the validation of XSLT documents with the help of an XSLT processor *that you can configure in the preferences pages* according to the XSLT version. For XSLT 1.0, the options are: Xalan, Saxon 6.5.5, Saxon 9.6.0.5 and a JAXP transformer specified by the main Java class. For XSLT 2.0, the options are: Saxon
9.6.0.5 and a JAXP transformer specified by the main Java class. For XSLT 3.0, the options are Saxon 9.6.0.5 and a JAXP transformer specified by the main Java class.

Custom Validation of XSLT Stylesheets

If you need to validate an XSLT stylesheet with a validation engine that is different from the built-in engine, you can configure external engines as custom XSLT validation engines in the Oxygen XML Developer plugin preferences. After a custom validation engine is properly configured, it can be applied on the current document by selecting it from the list of custom validation engines in the Validation toolbar drop-down list. The document is validated against the schema declared in the document.

There are two validators that are configured by default:

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

Associate a Validation Scenario

You can validate XSLT stylesheets by using a validation scenario. To create a validation scenario, select the Configure Validation Scenario(s) action from the XML menu, from the Validation toolbar drop-down list, or from the Validate submenu when invoking the contextual menu in the Navigator view.

You can validate an XSLT document using the engine defined in the transformation scenario, or a custom validation scenario. If you choose to validate using the engine from transformation scenario, and a transformation scenario is not associated with the current document or the engine has no validation support, the default engine is used. To set the default engine, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XSLT. The list of reusable scenarios for the appropriate document type is displayed. For more details go to Validation Scenario.

Editing XSLT Stylesheets in the Master Files Context

Smaller interrelated modules that define a complex stylesheet cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a function defined in a main stylesheet is not visible when you edit an included or imported module. Oxygen XML Developer plugin provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger stylesheet structure.

You can set a main XSLT stylesheet either using the master files support from the Navigator view, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Developer plugin warns you if the current module is not part of the dependencies graph computed for the main stylesheet. In this case, it considers the current module as the main stylesheet.

The advantages of editing in the context of main file include:

- Correct validation of a module in the context of a larger stylesheet structure.
- **Content Completion Assistant** displays all components valid in the current context.
- The Outline displays the components collected from the entire stylesheet structure.

To watch our video demonstration about editing XSLT stylesheets in the master files context, go to http://oxygenxml.com/demo/MasterFilesSupport.html.

Syntax Highlight

The XSL editor renders the CSS and JS scripts, and XPath expressions with dedicated coloring schemes. To customize the coloring schemes, open the Preferences dialog box and go to Editor > Syntax Highlight.
The items in the list of proposals offered by the **Content Completion Assistant** are context-sensitive. The proposed items are valid at the current caret position. You can enhance the list of proposals by specifying an additional schema. This schema is *defined by the user in the Content Completion / XSL preferences* page and can be: XML Schema, DTD, RELAX NG schema, or NVDL schema.

![Figure 52: XSLT Content Completion Window](image)

The **Content Completion Assistant** proposes numerous item types (such as templates, variables, parameters, keys, etc.) that are defined in the current stylesheet, and in the imported and included XSLT stylesheets. The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into stylesheets.

**Note:** For XSL and XSD resources, the **Content Completion Assistant** collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support, go to *Defining Master Files at Project Level*.

The extension functions built in the Saxon 6.5.5 and 9.6.0.5 transformation engines are presented in the content completion list only if the Saxon namespace ([http://saxon.sf.net](http://saxon.sf.net) for XSLT version 2.0 / 3.0 or [http://icl.com/saxon](http://icl.com/saxon) for XSLT version 1.0) is declared and one of the following conditions is true:

- the edited file has a transformation scenario that uses as transformation engine Saxon 6.5.5 (for XSLT version 1.0), Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE (for XSLT version 2.0 / 3.0).
- the edited file has a validation scenario that uses as validation engine Saxon 6.5.5 (for version 1.0), Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE (for version 2.0 / 3.0).
- the validation engine specified in **Options** page is Saxon 6.5.5 (for version 1.0), Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE (for version 2.0 / 3.0).

Additionally, the Saxon-CE-specific extension functions and instructions are presented in the Content Completion Assistant’s proposals list only if the [http://saxonica.com/ns/interactiveXSLT](http://saxonica.com/ns/interactiveXSLT) namespace is declared.

Namespace prefixes in the scope of the current context are presented at the top of the content completion window to speed up the insertion into the document of prefixed elements.
For the common namespaces like XSL namespace (http://www.w3.org/1999/XSL/Transform), XML Schema namespace (http://www.w3.org/2001/XMLSchema) or Saxon namespace (http://icl.com/saxon for version 1.0, http://saxon.sf.net/ for version 2.0/3.0), Oxygen XML Developer plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

Content Completion in XPath Expressions

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

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

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

Content completion for XPath expressions is started:

- On XPath operators detected in one of the match, select and test attributes of XSLT elements: ",', /, //, (, [,
  |, :, ::, $ symp
- For attribute value templates of non-XSLT elements, that is the \ character when detected as the first character of the attribute value.
- On request, if the combination Ctrl Space (Command Space on OS X) is pressed inside an edited XPath expression.

The items presented in the content completion window are dependent on:

- The context of the current XSLT element.
- The XML document associated with the edited stylesheet in the stylesheet transformation scenario.
- The XSLT version of the stylesheet (1.0, 2.0, or 3.0).

Note: The XSLT 3.0 content completion list of proposals includes specific elements and attributes for the 3.0 version.

For example, if the document associated with the edited stylesheet is:
If you enter an `xsl:template` element using the content completion assistant, the following actions are triggered:

- The `match` attribute is inserted automatically.
- The cursor is placed between the quotes.
- The XPath Content Completion Assistant automatically displays a popup window with all the XSLT axes, XPath functions and elements and attributes from the XML input document that can be inserted in the current context.

The set of XPath functions depends on the XSLT version declared in the root element `xsl:stylesheet`: 1.0, 2.0 or 3.0.

Figure 54: 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 55: 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.
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.

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.

The time delay configured in Preferences page for all content completion windows is applied also for the XPath expressions content completion window.
ToolTip Helper for the XPath Functions Arguments

When editing the arguments of an XPath/XSLT function, Oxygen XML Developer plugin tracks the current entered argument by displaying a tooltip containing the function signature. The currently edited argument is highlighted with a bolder font.

When moving the caret through the expression, the tooltip is updated to reflect the argument found at the caret position.

We want to concatenate the absolute values of two variables, named \( v_1 \) and \( v_2 \).

```
<xsl:template match="/">
  <xsl:value-of select="concat(abs($v1), abs($v2))"></xsl:value-of>
</xsl:template>
```

When moving the caret before the first \( \text{abs} \) function, Oxygen XML Developer plugin identifies it as the first argument of the \( \text{concat} \) function. The tooltip shows in bold font the following information about the first argument:

- Its name is \( \text{$arg1} \).
- Its type is \( \text{xdt:anyAtomicType} \).
- It is optional (note the \( ? \) sign after the argument type).

The function takes also other arguments, having the same type, and returns a \( \text{xs:string} \).

**Figure 59: XPath Tooltip Helper - Identify the \( \text{concat} \) Function's First Argument**

Moving the caret on the first variable \( \text{$v1} \), the editor identifies the \( \text{abs} \) as context function and shows its signature:

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

**Figure 60: XPath Tooltip Helper - Identify the \( \text{abs} \) Function's Argument**

Further, clicking the second \( \text{abs} \) function name, the editor detects that it represents the second argument of the \( \text{concat} \) function. The tooltip is repainted to display the second argument in bold font.

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

**Figure 61: XPath Tooltip Helper - Identify the \( \text{concat} \) Function's Second Argument**

The tooltip helper is available also in the XPath toolbar and the XPath Builder view.

The XSLT/XQuery Input View

The structure of the XML document associated to the edited XSLT stylesheet, or the structure of the source documents of the edited XQuery is displayed in a tree form in a view called XSLT/XQuery Input. The tree nodes represent the elements of the documents.
The XSLT Input View

If you click a node, the corresponding template from the stylesheet is highlighted. A node can be dragged from this view and dropped in the editor area for quickly inserting \texttt{xsl:template}, \texttt{xsl:for-each}, or other XSLT elements that have the \texttt{match/select/test} attribute already completed. The value of the attribute is the correct XPath expression that refers to the dragged tree node. This value is based on the current editing context of the drop spot.

For example, for the following XML document:

```xml
<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
<?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:template>
</xsl:stylesheet>
```

if you drag the \texttt{given} element and drop it inside the \texttt{xsl:for-each} element, the following popup menu is displayed:
The XSLT Input Drag and Drop Popup Menu

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

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

**Figure 64: XSLT Input Drag and Drop Result**

The XSLT Outline View

The XSLT Outline view displays the list of all the components (templates, attribute-sets, character-maps, variables, functions, keys, outputs) from both the edited stylesheet and its imports or includes. For XSL and XSD resources, the Outline view collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the Master Files support go to Defining Master Files at Project Level. To enable the Outline view, go to Window > Show View > Other > oXygen > Outline.

**Figure 65: The XSLT Outline View**

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

**Filter returns exact matches**

The text filter of the Outline view returns only exact matches;
Selection update on caret move
Controls the synchronization between Outline view and source document. The selection in the Outline view can be synchronized with the caret moves or the changes in the XSLT editor. Selecting one of the components from the Outline view also selects the corresponding item in the source document.

Show XML structure
Displays the XML document structure in a tree-like structure.

Show all components
Displays all components that were collected starting from the main file. This option is set by default.

Show only local components
Displays the components defined in the current file only.

Group by location/type
The stylesheet components can be grouped by location and type.

Show components
Shows the define patterns collected from the current document.

Flat presentation mode of the filtered results
When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions
Show/hide comments and processing instructions in the Outline view.

Show element name
Show/hide element name.

Show text
Show/hide additional text content for the displayed elements.

Show attributes
Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from the Outline preferences panel.

Configure displayed attributes
Displays the XML Structured Outline preferences page.

The following contextual menu actions are available:

Append Child
Displays a list of elements that can be inserted as children of the current element.

Insert Before
Displays a list of elements that can be inserted as siblings of the current element, before the current element.

Insert After
Displays a list of elements that can be inserted as siblings of the current element, after the current element.

Toggle Comment
Comments/uncomments the currently selected element.

Remove (Delete)
Removes the selected item from the stylesheet.

Search References Ctrl Shift R (Command Shift R on OS X)
Searches all references of the item found at current cursor position in the defined scope, if any. See Finding XSLT References and Declarations for more details.
Search References in...

Searches all references of the item found at current cursor position in the specified scope. See Finding XSLT References and Declarations for more details.

Component Dependencies

Allows you to see the dependencies for the current selected component. See Component Dependencies View for more details.

Rename Component in...

Renames the selected component. See XSLT Refactoring Actions for more details.

The stylesheet components information is presented on two columns: the first column presents the name and match attributes, the second column the mode attribute. If you know the component name, match or mode, you can search it in the Outline view by typing one of these pieces of information in the filter text field from the top of the view or directly on the tree structure. When you type the component name, match or mode in the text field, you can switch to the tree structure using:

- keyboard arrow keys
- Enter key
- Tab key
- Shift-Tab key combination

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

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

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

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

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

The Outline content and the editing area are synchronized. When you select a component in the Outline view, its definition is highlighted in the editing area.

Oxygen XML Developer plugin allows you to sort the components of the tree in the Outline view.

Note: Sorting groups in the Outline view is not supported.

Oxygen XML Developer plugin has a predefined order of the groups in the Outline view:

- for location, the names of the files are sorted alphabetically. The main file is the one you are editing and it is located at the top of the list
- for type, the order is: parameters, variables, templates, functions, set attributes, character-map

Note: When no grouping is available and the table is not sorted, Oxygen XML Developer plugin sorts the components depending on their order in the document. Oxygen XML Developer plugin also takes into account the name of the file that the components are part of.

XSLT Stylesheet Documentation Support

Oxygen XML Developer plugin offers built-in support for documenting XSLT stylesheets. If the expanded QName of the element has a non-null namespace URI, the xsl:stylesheet element may contain any element not from the XSLT namespace. Such elements are referenced as user-defined data elements. Such elements can contain the documentation for the stylesheet and its elements (top-level elements whose names are in the XSLT namespace). Oxygen XML Developer plugin offers its own XML schema that defines such documentation elements. The schema is named stylesheet_documentation.xsd and can be found in
When content completion is invoked inside an XSLT editor by pressing Ctrl Space (Command Space on OS X), it offers elements from the XSLT documentation schema (either the built-in one or one specified by user).

In Text mode, to add documentation blocks while editing use the Add component documentation action available in the contextual menu.

If the caret is positioned inside the `xsl:stylesheet` element context, documentation blocks are generated for all XSLT elements. If the caret is positioned inside a specific XSLT element (like a template or a function), a documentation block is generated for that element only.

Example of a documentation block using Oxygen XML Developer plugin built-in schema

```
<xd:doc>
  <xd:desc>
    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:desc>
  <xd:param name="string">
    String to be analyzed
  </xd:param>
  <xd:param name="searched">
    Marker string. Its last occurrence will be identified
  </xd:param>
  <xd:return>
    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:return>
</xd:doc>
```

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Developer plugin to generate detailed documentation in HTML format for the elements (top-level elements whose names are in the XSLT namespace) of an XSLT stylesheet. You are able to select what XSLT elements to include in the generated documentation and also the level of details to present for each of them. The elements are hyperlinked. To generate documentation in a custom format, other than HTML, you can edit the XSLT stylesheet used to generate the documentation, or create your own stylesheet.

To open the XSLT Stylesheet Documentation dialog box, select XSLT Stylesheet Documentation... from the XML Tools > Generate Documentation menu or from the Generate Stylesheet Documentation action from the contextual menu of the Navigator view.
Figure 66: The Output Panel of the XSLT Stylesheet Documentation Dialog Box

The **XSL URL** field of the dialog box must contain the full path to the XSL Stylesheet file you want to generate documentation for. The stylesheet can be either a local or a remote one. You can also specify the path of the stylesheet using editor variables.

You can choose to split the output into multiple files using different split criteria. For large XSLT stylesheets being documented, choosing a different split criterion may generate smaller output files providing a faster documentation browsing.

The available split criteria are:

- **by location** - Each output file contains the XSLT elements from the same stylesheet.
- **by namespace** - Each output file contains information about elements with the same namespace.
- **by component** - Each output file contains information about one stylesheet XSLT element.

Figure 67: The Settings Panel of the XSLT Stylesheet Documentation Dialog Box
When you generate documentation for an XSLT stylesheet you can choose what XSLT elements to include in the output (templates, functions, global parameters, global variables, attribute sets, character maps, keys, decimal formats, output formats, XSLT elements from referenced stylesheets) and the details to include in the documentation:

- **Documentation** - Shows the documentation for each XSLT element. For HTML format, the user-defined data elements that are recognized and transformed in documentation blocks of the XSLT elements they precede, are the ones from the following schemas:
  - Oxygen XML Developer plugin built-in XSLT documentation schema.
  - A subset of DocBook 5 elements. The recognized elements are: `section`, `sect1` to `sect5`, `emphasis`, `title`, `ulink`, `programlisting`, `para`, `orderedlist`, `itemizedlist`.
  - A subset of DITA elements. The recognized elements are: `concept`, `topic`, `task`, `codeblock`, `p`, `b`, `i`, `ul`, `ol`, `pre`, `sl`, `sli`, `step`, `steps`, `li`, `title`, `xref`.
  - Full XHTML 1.0 support.
  - XSLStyle documentation environment. XSLStyle uses DocBook or DITA languages inside its own user-defined data elements. The supported DocBook and DITA elements are the ones mentioned above.
  - Doxsl documentation framework. Supported elements are: `codefrag`, `description`, `para`, `docContent`, `documentation`, `parameter`, `function`, `docSchema`, `link`, `list`, `listitem`, `module`, `parameter`, `template`, `attribute-set`;

Other XSLT documentation blocks that are not recognized will just be serialized inside an HTML `pre` element. You can change this behavior by using a custom format instead of the built-in HTML format and providing your own XSLT stylesheets.

- **Use comments** - Controls whether the comments that precede an XSLT element is treated as documentation for the element they precede. Comments that precede or succeed the `xsl:stylesheet` element, are treated as documentation for the whole stylesheet. Please note that comments that precede an import or include directive are not collected as documentation for the imported/included module. Also comments from within the body of the XSLT elements are not collected at all.
- **Namespace** - Shows the namespace for named XSLT elements.
- **Location** - Shows the stylesheet location for each XSLT element.
- **Parameters** - Shows parameters of templates and functions.
- **References** - Shows the named XSLT elements that are referenced from within an element.
- **Used by** - Shows the list of all the XSLT elements that reference the current named element.
- **Supersedes** - Shows the list of all the XSLT elements that are superseded the current element.
- **Overriding** - Shows the list of all the XSLT elements that override the current element.
- **Return type** - Shows the return type of the function.
- **Source** - Shows the text stylesheet source for each XSLT element.
- **Import precedence** - Shows the computed import precedence as declared in XSL transformation specifications.
- **Generate index** - Creates an index with all the XSLT elements included in the documentation.
- **Load settings / Export settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the Export settings button, and reloaded when necessary with the Load settings button.

**Generate Documentation in HTML Format**

The generated documentation looks like:
Figure 68: XSLT Stylesheet Documentation Example

The generated documentation includes the following:

- Table of Contents - You can group the contents by namespace, location, or component type. The XSLT elements from each group are sorted alphabetically (named templates are presented first and the match ones second).
- Information about main, imported, and included stylesheets - This information consists of:
  - XSLT modules included or imported by the current stylesheet
  - the XSLT stylesheets where the current stylesheet is imported or included
  - the stylesheet location

Figure 69: 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 70: 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 71: Documentation for an XSLT Element
Generate Documentation in a Custom Format

XSLT stylesheet documentation can be also generated in a custom format. You can choose the format from the XSLT Stylesheet Documentation dialog box. Specify your own stylesheet to transform the intermediary XML generated in the documentation process. You must write your stylesheet based on the schema xslDocSchema.xsd from [OXYGEN_DIR]/frameworks/stylesheet_documentation. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in [OXYGEN_DIR]/frameworks/stylesheet_documentation/xsl.

![Custom Format Options Dialog Box](image)

Figure 72: The Custom Format Options Dialog Box

When using a custom format, you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating Documentation From the Command Line Interface

You can export the settings of the XSLT Stylesheet Documentation dialog box to an XML file by pressing the Export settings button. With the exported settings file, you can generate the same documentation from the command line by running the script stylesheetDocumentation.bat (on Windows)/stylesheetDocumentation.sh (on OS X / Unix / Linux) located in the Oxygen XML Developer plugin installation folder. The script can be integrated in an external batch process launched from the command-line interface.

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

Example of an XML Configuration File

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

The following search actions related with XSLT references and declarations are available from the Search submenu of the contextual menu:
Search References (Also available from the XSL menu) - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined but the currently edited resource is not part of the range of determined resources, a warning dialog box is displayed that allows you to define another search scope.

Search References in... - Searches all references of the item found at current cursor position in the file or files that you specify when a scope is defined.

Search Declarations (Also available from the XSL menu) - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined but the currently edited resource is not part of the range of resources determined by this scope, a warning dialog box is displayed that allows you to define another search scope.

Search Declarations in... - Searches all declarations of the item found at current cursor position in the file or files that you specify when a scope is defined.

Search Occurrences in File - Searches all occurrences of the item at the caret position in the currently edited file.

The following action is available from the XSL menu:

Show Definition - Moves the cursor to the location of the definition of the current item.

Note: You can also use the Ctrl Click (Command Click on OS X) shortcut on a reference to display its definition.

Highlight Component Occurrences

When a component (for example variable or named template) is found at current cursor position, Oxygen XML Developer plugin performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document.

Note: Oxygen XML Developer plugin also supports occurrences highlight for template modes.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is enabled by default. To configure it, open the Preferences dialog box and go to Editor > Mark Occurrences. A search can also be triggered with the Search > Search Occurrences in File (Ctrl Shift U (Command Shift U on OS X)) contextual menu action. Matches are displayed in separate tabs of the Results view.

XSLT Refactoring Actions

Oxygen XML Developer plugin offers a set of actions that allow changing the structure of an XSLT stylesheet without changing the results of running it in an XSLT transformation. Depending on the selected text, the following refactoring actions are available from Refactoring submenu from the contextual menu of the current editor:

Extract template... - Extracts the selected XSLT instructions sequence into a new template. Opens a dialog that allows you to specify the name of the new template to be created. The possible changes to perform on the document can be previewed before altering the document. After pressing OK, the template is created and the selection is replaced with a <xsl:call-template> instruction referencing the newly created template.

Note: This action is available only when the selection contains well-formed elements.

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

Move to another stylesheet... - Allows you to move one or more XSLT global components (templates, functions or parameters) to another stylesheet. Active only when these components are selected. Follow these steps:

execute the Move to another stylesheet action. You will be prompted to select the destination stylesheet, which can be a new stylesheet or an already existing one.
• press the Choose button to navigate to the destination stylesheet file. Oxygen XML Developer plugin will automatically check if the destination stylesheet is already contained by the hierarchy of the current stylesheet. If it is not contained, choose if the destination stylesheet will be referenced (imported or included) or not from the current stylesheet. The following options are available:
  • Include - the current stylesheet will use an xsl:include instruction to reference the destination stylesheet.
  • Import - the current stylesheet will use an xsl:import instruction to reference the destination stylesheet.
  • None - there will be created no relation between the current and destination styleheets.

• press the Move button to move the components to the destination stylesheet. After the action's execution, the moved components are highlighted in the destination stylesheet.

• Convert attributes to xsl:attributes - Converts the attributes from the selected element and represents each of them with an xsl:attribute instruction. For example, from the following element:

```xml
<person id="Big{test}Boss"/>
```

you obtain:

```xml
<person>
  <xsl:attribute name="id">
    <xsl:text>Big</xsl:text>
    <xsl:value-of select="test"/>
    <xsl:text>Boss</xsl:text>
  </xsl:attribute>
</person>
```

• Convert xsl:if into xsl:choose/xsl:when - Converts an xsl:if block to an xsl:when block surrounded by an xsl:choose element. For example, the following block:

```xml
<xsl:if test="a">
  <!-- XSLT code -->
</xsl:if>
```

is converted to:

```xml
<xsl:choose>
  <xsl:when test="a">
    <!-- XSLT code -->
  </xsl:when>
  <xsl:otherwise>
    <!-- XSLT code -->
  </xsl:otherwise>
</xsl:choose>
```

where the | character is the current caret position.

• Extract local variable - Allows you to create a new local variable by extracting the selected XPath expression. After creating the new local variable before the current element, Oxygen XML Developer plugin allows you to edit in-place the variable's name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.

• Extract global variable - Allows you to create a new global variable by extracting the selected XPath expression. After creating the new global variable, Oxygen XML Developer plugin allows you to edit in-place the variable's name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.

  Note: Oxygen XML Developer plugin checks if the selected expression depends on local variables or parameters that are not available in the global context where the new variable is created.

• Extract template parameter - Allows you to create a new template parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Developer plugin allows you to edit in-place its name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.
• **Extract global parameter** - Allows you to create a new global parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Developer plugin allows you to edit in-place its name.

  **Note:** The action is active on a selection made inside an attribute that contains an XPath expression.

  **Note:** Oxygen XML Developer plugin checks if the selected expression depends on local variables or parameters that are not available in the global context where the new parameter is created.

• **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.

  **Note:** These refactoring actions are also proposed by the **Quick Assist support**.

  To watch our video demonstration about XSLT refactoring, go to [http://oxygenxml.com/demo/XSL_Refactoring.html](http://oxygenxml.com/demo/XSL_Refactoring.html).

**XSLT Resource Hierarchy/Dependencies View**

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a stylesheet. To open this view, go to **Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies**.

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

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

- **Refresh**
  - Refreshes the Hierarchy/Dependencies structure.

- **Stop**
  - Stops the hierarchy/dependencies computing.
Show Hierarchy
Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies
Allows you to choose a resource to compute the dependencies structure.

Configure
Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History
Provides access to the list of previously computed dependencies. Use the Clear history button to remove all items from this list.

The contextual menu contains the following actions:

Open
Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location
Copies the location of the resource.

Move resource
Moves the selected resource.

Rename resource
Renames the selected resource.

Show Resource Hierarchy
Shows the hierarchy for the selected resource.

Show Resource Dependencies
Shows the dependencies for the selected resource.

Add to Master Files
Adds the currently selected resource in the Master Files directory.

Expand All
Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All
Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

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

Moving/Renaming XSLT Resources
You are able to move and rename a resource presented in the Resource/Hierarchy Dependencies view, using the Rename resource and Move resource refactoring actions from the contextual menu.

When you select the Rename action in the contextual menu of the Resource/Hierarchy Dependencies view, the Rename resource dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the Move action from the contextual menu of the Resource/Hierarchy Dependencies view, the Move resource dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
• **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

## Component Dependencies View

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

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

![Component Dependencies View - Hierarchy for table.xsl](image)

**Figure 76: Component Dependencies View - Hierarchy for table.xsl**

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

- **Refresh**
  
  Refreshes the dependencies structure.

- **Stop**
  
  Stops the dependencies computing.

- **Configure**
  
  Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

- **History**
  
  Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:
Go to First Reference
Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component
Shows the definition of the current selected component in the dependencies tree.

Tip: If a component contains multiple references to another, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon ☰.

XSLT Quick Assist Support
The Quick Assist support helps you to rapidly access search and refactoring actions. If one or more actions are available in the current context, they are accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the Quick Assist menu if you press Ctrl + 1 keys (Meta 1 on Mac OS X) on your keyboard.

Two categories of actions are available in the Quick Assist menu:

• Actions available on a selection made inside an attribute that contains an XPath expression:

  * Extract template - Extracts the selected XSLT instructions sequence into a new template.
  * Move to another stylesheet - Allows you to move one or more XSLT global components (templates, functions or parameters) to another stylesheet.
  * Extract local variable - Allows you to create a new local variable by extracting the selected XPath expression.
  * Extract global variable - Allows you to create a new global variable by extracting the selected XPath expression.
  * Extract template parameter - Allows you to create a new template parameter by extracting the selected XPath expression.
  * Extract global parameter - Allows you to create a new global parameter by extracting the selected XPath expression.

![Figure 77: XSLT Quick Assist Support - Refactoring Actions](image)

• actions available when the cursor is positioned over the name of a component:

  * Rename Component in... Renames the component and all its dependencies.
  * Search Declarations Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.
  * Search References Searches all references of the component in a predefined scope.
  * Component Dependencies Searches the component dependencies in a predefined scope.
Change Scope...
Configures the scope that will be used for future search or refactor operations.

Rename Component
Allows you to rename the current component in-place.

Search Occurrences
Searches all occurrences of the component within the current file.

![Figure 78: XSLT Quick Assist Support - Component Actions](image)

**XSLT Quick Fix Support**

The Oxygen XML Developer plugin Quick Fix support helps you resolve different errors that appear in a stylesheet by offering quick fixes to problems like a missing template, misspelled template name, missing function or references to an undeclared variable or parameter.

![Figure 79: XSLT Functions Quick Fix](image)

To activate the feature, when Oxygen XML Developer plugin finds a validation error in an XSLT stylesheet, place the caret in the highlighted area of text. If Oxygen XML Developer plugin can provide a quick fix for that error, the `галка` icon is displayed in the left side stripe. When you click this icon, the list of available fixes is displayed. Also, you can invoke the quick fix menu if you press Ctrl + 1 on your keyboard.

**Note:** The quick fixes are available only when validating an XSLT file with Saxon HE/PE/EE.

![Figure 80: XSLT Variables and Parameters Quick Fix](image)
Oxygen XML Developer plugin provides quick fixes for the following cases:

- **Template does not exist**, when the template name referenced in a `call-template` element does not exist. The following fixes are available:
  - **Create template "templateName"** - creates a template and generates its corresponding parameters. The template name and parameter names and types are collected from the `call-template` element.
  - **Change reference to "newTemplateReference"** - changes the name of the missing template referenced in the `call-template` element. The proposed new names are the existing templates with names similar with the missing one.

- **Variable/Parameter not declared**, when a parameter or variable reference cannot be found. The following fixes are available:
  - **Create global variable "varName"** - creates a global variable with the specified name in the current stylesheet. The new variable is added at the beginning of the stylesheet after the last global variable or parameter declaration.
  - **Create global parameter "paramName"** - creates a global parameter with the specified name in the current stylesheet. The new parameter is added at the beginning of the stylesheet after the last global parameter or variable declaration.
  - **Create local variable "varName"** - creates a local variable with the specified name before the current element.
  - **Create template parameter "paramName"** - creates a new parameter with the specified name in the current template. This fix is available if the error is located inside a template.
  - **Create function parameter "paramName"** - creates a new parameter with the specified name in the current function. This fix is available if the error is located inside a function.
  - **Change reference to "varName"** - changes the name of the referenced variable/parameter to an existing local or global variable/parameter, that has a similar name with the current one.

- **Parameter from a called template is not declared**, when a parameter referenced from a `call-template` element is not declared. The following fixes are available:
  - **Create parameter "paramName" in the template "templateName"** - creates a new parameter with the specified name in the referenced template.
  - **Change "paramName" parameter reference to "newParamReference"** - changes the parameter reference from the `call-template` element to a parameter that is declared in the called template.
  - **Remove parameter "paramName" from call-template** - removes the parameter with the specified name from the `call-template` element.

- **No value supplied for required parameter**, when a required parameter from a template is not referenced in a `call-template` element. The following quick-fix is available:
  - **Add parameter "paramName" in call-template** - creates a new parameter with the specified name in call-template element.

- **Function "prefix:functionName()" has not been defined**, when a function declaration is not found. The following quick fixes are available:
  - **Create function "prefix:functionName(param1, param2)"** - creates a new function with the specified signature, after the current top level element from stylesheet.
  - **Change function to "newFunctionName(...)"** - changes the referenced function name to an already defined function. The proposed names are collected from functions with similar names and the same number of parameters.

- **Attribute-set "attrSetName" does not exist**, when the referenced attribute set does not exist. The following quick fixes are available:
  - **Create attribute-set "attrSetName"** - creates a new attribute set with the specified name, after the current top level element from stylesheet.
  - **Change reference to "attrSetName"** - changes the referenced attribute set to an already defined one.

- **Character-map "chacterMap" has not been defined**, when the referenced character map declaration is not found. The following quick fixes are available:
Create character-map "characterMapName" - creates a new character map with the specified name, after the current top level element from stylesheet.

Change reference to "characterMapName" - changes the referenced character map to an already defined one.

XSLT Unit Test (XSpec)

XSpec is a behavior driven development (BDD) framework for XSLT and XQuery. XSpec consists of a syntax for describing the behavior of your XSLT or XQuery code, and some code that enables you to test your code against those descriptions.

To create an XSLT Unit Test, go to File > New > XSLT Unit Test. You can also create an XSLT Unit Test from the contextual menu of an XSL file in the Project view. Oxygen XML Developer plugin allows you to customize the XSpec document when you create it. In the customization dialog, you can enter the path to an XSL document or to a master XSL document.

To run an XSLT Unit Test, open the XSPEC file in an editor and click Apply Transformation Scenario(s) on the main toolbar.

Note: The transformation scenario is defined in the XSPEC document type.

When you create an XSpec document based on an XSL document, Oxygen XML Developer plugin uses information from the validation and transformation scenarios associated with the XSL file. From the transformation scenario Oxygen XML Developer plugin uses extensions and properties of Saxon 9.6.0.5, improving the ANT scenario associated with the XSpec document.

An XSpec file contains one, or more test scenarios. You can test a stylesheet in one of the following ways:

• test an entire stylesheet;

Testing is performed in a certain context. You can define a context as follows:
• inline context - building the test based on a string:

```xml
<x:scenario label="when processing a para element">
  <x:context>
    <para>...</para>
  </x:context>
  ...
</x:scenario>
```

• based on an external file, or on a part of an external file extracted with an XPath expression.

```xml
<x:scenario label="when processing a para element">
  <x:context href="source/test.xml" select="/doc/body/p[1]" />
  ...
</x:scenario>
```

• test a function:

```xml
<x:scenario label="when capitalising a string">
  <x:call function="eg:capital-case">
    <x:param select="'an example string'" />
    <x:param select="true()" />
  </x:call>
  ...
</x:scenario>
```

• test a template with a name.

```xml
<x:call template="createTable">
  <x:param name="nodes">
    <value>A</value>
    <value>B</value>
  </x:param>
  <x:param name="cols" select="2" />
</x:call>
```

You are able to reference test files between each other, which allows you to define a suite of tests. For further details about test scenarios, go to [http://code.google.com/p/xspec/wiki/WritingScenarios](http://code.google.com/p/xspec/wiki/WritingScenarios).

## Editing XML Schemas

An XML Schema describes the structure of an XML document and is used to validate XML document instances against it, to check that the XML instances conform to the specified requirements. If an XML instance conforms to the schema then it is said to be valid, otherwise it is invalid.

Two editing modes are provided for working with XML Schema:

- The **Text** editing mode.
- The visual **Design** editing mode.

Note: Oxygen XML Developer plugin offers support for both XML schema 1.0 and 1.1.

### XML Schema Diagram Editing Mode

This section explains how to use the graphical diagram of a W3C XML Schema.

**Introduction**

XML Schemas enable document designers to specify the allowed structure and content of an XML document and to check if an XML document is valid.

Oxygen XML Developer plugin provides a simple and expressive **Design** mode for editing XML Schemas. The schema diagram helps both the content authors who want to understand a schema and schema designers who develop complex schemas.
To watch our video demonstration about the basic aspects of designing an XML Schema using the new Schema Editor, go to [http://oxygenxml.com/demo/XML_Schema_Editing.html](http://oxygenxml.com/demo/XML_Schema_Editing.html).

**XML Schema Components**

A schema diagram contains a series of interconnected components. To quickly identify the relation between two connected components, the connection is represented as:

- A thick line to identify a connection with a required component (in the following image, *family* is a required element).
• A thin line to identify a connection with an optional component (in the following image, email is an optional element).

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

**xs:schema**

```
  <xs:schema
    targetNamespace=http://www.oxygenxml.com/supported-grammars/>
```

Defines the root element of a schema. A schema document contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-schema](http://www.w3.org/TR/xmlschema11-1/#element-schema).

By default it displays the `targetNamespace` property when rendered.

**xs:schema properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Namespace</td>
<td>The schema target namespace.</td>
<td>Any URI</td>
</tr>
<tr>
<td>Element Form Default</td>
<td>Determining whether local element declarations will be namespace-qualified by default.</td>
<td>qualified, unqualified, [Empty]. Default value is unqualified.</td>
</tr>
<tr>
<td>Attribute Form Default</td>
<td>Determining whether local attribute declarations will be namespace-qualified by default.</td>
<td>qualified, unqualified, [Empty]. Default value is unqualified.</td>
</tr>
<tr>
<td>Block Default</td>
<td>Default value of the <code>block</code> attribute of <code>xs:element</code> and <code>xs:complexType</code>.</td>
<td>#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, restriction extension substitution, restriction extension substitution, restriction extension substitution, [Empty].</td>
</tr>
<tr>
<td>Final Default</td>
<td>Default value of the <code>final</code> attribute of <code>xs:element</code> and <code>xs:complexType</code>.</td>
<td>#all, restriction, extension, restriction extension, [Empty].</td>
</tr>
<tr>
<td>Default Attributes</td>
<td>Specifies a set of attributes that apply to every complex Type in a schema document.</td>
<td>Any.</td>
</tr>
<tr>
<td>Xpath Default Namespace</td>
<td>The default namespace used when the XPath expression is evaluated.</td>
<td><code>##defaultNamespace, ##targetNamespace, ##local</code>.</td>
</tr>
<tr>
<td>Version</td>
<td>Schema version</td>
<td>Any token.</td>
</tr>
<tr>
<td>ID</td>
<td>The schema id</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>SystemID</td>
<td>The schema system id</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>
**xs:element**

Defines an element. An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions. See more info at http://www.w3.org/TR/xmlschema11-1/#element-element.

An element by default displays the following properties when rendered in the diagram: `default, fixed, abstract and type`. When referenced or declared locally, the element graphical representation also contains the value for the `minOccurs` and `maxOccurs` properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the element are drawn using dotted lines if the element is optional.

### xs:element properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Mentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The element name. Always required.</td>
<td>Any NCName for global or local elements, any QName for element references.</td>
<td>If missing, will be displayed as ['element'] in diagram.</td>
</tr>
<tr>
<td>Is Reference</td>
<td>When set, the local element is a reference to a global element.</td>
<td>true/false</td>
<td>Appears only for local elements.</td>
</tr>
<tr>
<td>Type</td>
<td>The element type.</td>
<td>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].</td>
<td>For all elements. For references, the value is set in the referenced element.</td>
</tr>
<tr>
<td>Base Type</td>
<td>The extended/restricted base type.</td>
<td>All declared or built-in types</td>
<td>For elements with complex type, with simple or complex content.</td>
</tr>
<tr>
<td>Mixed</td>
<td>Defines if the complex type content model will be mixed.</td>
<td>true/false</td>
<td>For elements with complex type.</td>
</tr>
<tr>
<td>Content</td>
<td>The content of the complex type.</td>
<td>simple/complex</td>
<td>For elements with complex type which extends/restricts a base type. It is automatically detected.</td>
</tr>
<tr>
<td>Content Mixed</td>
<td>Defines if the complex content model will be mixed.</td>
<td>true/false</td>
<td>For elements with complex type which has a complex content.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
<td>Possible Values</td>
<td>Mentions</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------------</td>
<td>---------</td>
</tr>
<tr>
<td>Default</td>
<td>Default value of the element. A default value is automatically assigned to the element when no other value is specified.</td>
<td>Any string</td>
<td>The fixed and default attributes are mutually exclusive.</td>
</tr>
<tr>
<td>Fixed</td>
<td>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.</td>
<td>Any string</td>
<td>The fixed and default attributes are mutually exclusive.</td>
</tr>
<tr>
<td>Min Occurs</td>
<td>Minimum number of occurrences of the element.</td>
<td>A numeric positive value. Default value is 1</td>
<td>Only for references/local elements</td>
</tr>
<tr>
<td>Max Occurs</td>
<td>Maximum number of occurrences of the element.</td>
<td>A numeric positive value. Default value is 1</td>
<td>Only for references/local elements</td>
</tr>
<tr>
<td>Substitution Group</td>
<td>Qualified name of the head of the substitution group to which this element belongs.</td>
<td>All declared elements. For XML Schema 1.1 this property supports multiple values.</td>
<td>For global and reference elements</td>
</tr>
<tr>
<td>Abstract</td>
<td>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.</td>
<td>true/false</td>
<td>For global elements and element references</td>
</tr>
<tr>
<td>Form</td>
<td>Defines if the element is &quot;qualified&quot; (i.e., belongs to the target namespace) or &quot;unqualified&quot; (i.e., doesn't belong to any namespace).</td>
<td>unqualified/qualified</td>
<td>Only for local elements</td>
</tr>
<tr>
<td>Nillable</td>
<td>When this attribute is set to true, the element can be declared as nil using an xsi:nil attribute in the instance documents.</td>
<td>true/false</td>
<td>For global elements and element references</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
<td>Possible Values</td>
<td>Mentions</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Target Namespace</td>
<td>Specifies the target namespace for local element and attribute declarations. The namespace URI may be different from the schema target namespace. This property is available for local elements only.</td>
<td>Not editable property.</td>
<td>For all elements.</td>
</tr>
<tr>
<td>Block</td>
<td>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'/restitution' 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.</td>
<td>#all, restriction, extension, substitution, extension restriction, restriction substitution, restriction extension substitution</td>
<td>For global elements and element references</td>
</tr>
<tr>
<td>Final</td>
<td>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.</td>
<td>#all, restriction, extension, restriction extension, [Empty]</td>
<td>For global elements and element references</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any id</td>
<td>For all elements.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
<td>Possible Values</td>
<td>Mentions</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
<td>For all elements.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property.</td>
<td>For all elements.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
<td>For all elements.</td>
</tr>
</tbody>
</table>

**xs:attribute**

Defines an attribute. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-attribute](http://www.w3.org/TR/xmlschema11-1/#element-attribute).

An attribute by default displays the following properties when rendered in the diagram: default, fixed, use and type. Connectors to the attribute are drawn using dotted lines if the attribute use is optional. The attribute name is stroked out if prohibited.

**xs:attribute properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Value</th>
<th>Mentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Attribute name. Always required.</td>
<td>Any NCName for global/local attributes, all declared attributes' QName for references.</td>
<td>For all local or global attributes. If missing, will be displayed as '[attribute]' in the diagram.</td>
</tr>
<tr>
<td>Is Reference</td>
<td>When set, the local attribute is a reference.</td>
<td>true/false</td>
<td>For local attributes.</td>
</tr>
<tr>
<td>Type</td>
<td>Qualified name of a simple type.</td>
<td>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.</td>
<td>For all attributes. For references, the type is set to the referenced attribute.</td>
</tr>
<tr>
<td>Default</td>
<td>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.</td>
<td>Any string</td>
<td>For all local or global attributes. For references the value is from the referenced attribute.</td>
</tr>
<tr>
<td>Fixed</td>
<td>When specified, the value of the attribute is fixed and must be equal to this value. The default and fixed attributes are mutually exclusive.</td>
<td>Any string</td>
<td>For all local or global attributes. For references the value is from the referenced attribute.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
<td>Possible Value</td>
<td>Mentions</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Use</td>
<td>Possible usage of the attribute. Marking an attribute &quot;prohibited&quot; is useful to exclude attributes during derivations by restriction.</td>
<td>optional, required, prohibited</td>
<td>For local attributes</td>
</tr>
<tr>
<td>Form</td>
<td>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.</td>
<td>unqualified/qualified</td>
<td>For local attributes.</td>
</tr>
<tr>
<td>Inheritable</td>
<td>Specifies if the attribute is inheritable. Inheritable attributes can be used by &lt;alternative&gt; element on descendant elements.</td>
<td>true/false</td>
<td>For all local or global attributes. The default value is false. This property is available for XML Schema 1.1.</td>
</tr>
<tr>
<td>Target Namespace</td>
<td>Specifies the target namespace for local attribute declarations. The namespace URI may be different from the schema target namespace.</td>
<td>Any URI</td>
<td>Setting a target namespace for local attribute is useful only when restricts attributes of a complex type that is declared in other schema with different target namespace. This property is available for XML Schema 1.1.</td>
</tr>
</tbody>
</table>

For all attribute groups:

- **ID**: The component id. Any id. For all attributes.
- **Component**: The edited component name. Not editable property. For all attributes.
- **Namespace**: The component namespace. Not editable property. For all attributes.
- **System ID**: The component system id. Not editable property. For all attributes.

### xs:attributeGroup

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

**xs:attributeGroup properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Mentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Attribute group name. Always required.</td>
<td>Any NCName for global attribute groups, all declared attribute groups for reference.</td>
<td>For all global or referenced attribute groups. If missing, will be displayed as '[attributeGroup]' in diagram.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any id</td>
<td>For all attribute groups.</td>
</tr>
</tbody>
</table>
**xs:complexType**

Defines a top level complex type. Complex Type Definitions provide for: See more data at [http://www.w3.org/TR/xmlschema11-1/#element-complexType](http://www.w3.org/TR/xmlschema11-1/#element-complexType).

- Constraining element information items by providing Attribute Declarations governing the appearance and content of attributes.
- Constraining element information item children to be empty, or to conform to a specified element-only or mixed content model, or else constraining the character information item children to conform to a specified simple type definition.
- Using the mechanisms of Type Definition Hierarchy to derive a complex type from another simple or complex type.
- Specifying post-schema-validation infoset contributions for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

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

### xs:complexType properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Mentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the complex type. Always required.</td>
<td>Any NCName</td>
<td>Only for global complex types. If missing, will be displayed as '[complexType]' in diagram.</td>
</tr>
<tr>
<td>Base Type Definition</td>
<td>The name of the extended/restricted types.</td>
<td>Any from the declared simple or complex types.</td>
<td>For complex types with simple or complex content.</td>
</tr>
<tr>
<td>Derivation Method</td>
<td>The derivation method.</td>
<td>restriction/extension</td>
<td>Only when base type is set. If the base type is a simple type, the derivation method is always extension.</td>
</tr>
<tr>
<td>Content</td>
<td>The content of the complex type.</td>
<td>simple/complex</td>
<td>For complex types which extend/restrict a base type. It is automatically detected.</td>
</tr>
<tr>
<td>Content Mixed</td>
<td>Specifies if the complex content model will be mixed.</td>
<td>true/false</td>
<td>For complex contents.</td>
</tr>
<tr>
<td>Mixed</td>
<td>Specifies if the complex type content model will be mixed.</td>
<td>true/false</td>
<td>For global and anonymous complex types.</td>
</tr>
<tr>
<td>Abstract</td>
<td>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.</td>
<td>true/false</td>
<td>For global and anonymous complex types.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
<td>Possible Values</td>
<td>Mentions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Block</td>
<td>Controls whether a substitution (either through 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 &quot;unblock&quot; them), which can also be blocked in the element definition. The default value is defined by the blockDefault attribute of xs:schema.</td>
<td>all, extension, restriction, extension restriction, [Empty]</td>
<td>For global complex types.</td>
</tr>
<tr>
<td>Final</td>
<td>Controls whether the complex type can be further derived by extension or restriction to create new complex types.</td>
<td>all, extension, restriction, extension restriction, [Empty]</td>
<td>For global complex types.</td>
</tr>
<tr>
<td>Default Attributes Apply</td>
<td>The schema element can carry a defaultAttributes attribute, which identifies an attribute group. Each complexType defined in the schema document then automatically includes that attribute group, unless this is overridden by the defaultAttributesApply attribute on the complexType element.</td>
<td>true/false</td>
<td>This property is available only for XML Schema 1.1.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any id</td>
<td>For all complex types.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property</td>
<td>For all complex types.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property</td>
<td>For all complex types.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property</td>
<td>For all complex types.</td>
</tr>
</tbody>
</table>

**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/xmlschema11-1/#element-simpleType](http://www.w3.org/TR/xmlschema11-1/#element-simpleType).

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

**xs:simpleType properties**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Simple type name. Always required.</td>
<td>Any NCName.</td>
<td>Only for global simple types. If missing, will be displayed as '[simpleType]' in diagram.</td>
</tr>
<tr>
<td>Derivation</td>
<td>The simple type category: restriction, list or union.</td>
<td>restriction, list or union</td>
<td>For all simple types.</td>
</tr>
<tr>
<td>Base Type</td>
<td>A simple type definition component. Required if derivation method is set to restriction.</td>
<td>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.</td>
<td>For global and anonymous simple types with the derivation method set to restriction.</td>
</tr>
<tr>
<td>Item Type</td>
<td>A simple type definition component. Required if derivation method is set to list.</td>
<td>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.</td>
<td>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).</td>
</tr>
<tr>
<td>Member Types</td>
<td>Category for grouping union members.</td>
<td>Not editable property.</td>
<td>For global and anonymous simple types with the derivation method set to union.</td>
</tr>
<tr>
<td>Member</td>
<td>A simple type definition component. Required if derivation method is set to union.</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Final</td>
<td>Blocks any further derivations of this datatype</td>
<td>#all, list, restriction, union, list restriction, list union,</td>
<td>Only for global simple types.</td>
</tr>
</tbody>
</table>
### xs:alternative

The `type alternatives` mechanism allows you to specify type substitutions on an element declaration.

**Note:** `xs:alternative` is available for XML Schema 1.1.

#### xs:alternative properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Specifies type substitutions for an element, depending on the value of the attributes.</td>
<td>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].</td>
</tr>
<tr>
<td>Test</td>
<td>Specifies an XPath expression. If the XPath condition is valid, the specified type is selected as the element type. The expressions allowed are limited to a subset of XPath 2.0. Only the attributes of the current element and inheritable attributes from ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.</td>
<td>An XPath expression.</td>
</tr>
<tr>
<td>XPath Default Namespace</td>
<td>The default namespace used when the XPath expression is evaluated.</td>
<td><code>##defaultNamespace</code>, <code>##targetNamespace</code>, <code>##local</code>.</td>
</tr>
<tr>
<td>ID</td>
<td>Specifies the component ID.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>Specifies the type of XML schema component.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>Points to the document location of the schema.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>
**xs:group**

Defines a group of elements to be used in complex type definitions. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-group](http://www.w3.org/TR/xmlschema11-1/#element-group).

When referenced, the graphical representation also contains the value for the `minOccurs` and `maxOccurs` properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the group are drawn using dotted lines if the group is optional.

**xs:group properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Mentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The group name. Always required.</td>
<td>Any NCName for global groups, all declared groups for reference.</td>
<td>If missing, will be displayed as '[group]' in diagram.</td>
</tr>
<tr>
<td>Min Occurs</td>
<td>Minimum number of occurrences of the group.</td>
<td>A numeric positive value. Default value is 1.</td>
<td>Appears only for reference groups.</td>
</tr>
<tr>
<td>Max Occurs</td>
<td>Maximum number of occurrences of the group.</td>
<td>A numeric positive value. Default value is 1.</td>
<td>Appears only for reference groups.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any id</td>
<td>For all groups.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
<td>For all groups.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property</td>
<td>For all groups.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
<td>For all groups.</td>
</tr>
</tbody>
</table>

**xs:include**

Adds multiple schemas with the same target namespace to a document. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-include](http://www.w3.org/TR/xmlschema11-1/#element-include).

**xs:include properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Location</td>
<td>Included schema location.</td>
<td>Any URI</td>
</tr>
<tr>
<td>ID</td>
<td>Include ID.</td>
<td>Any ID</td>
</tr>
<tr>
<td>Component</td>
<td>The component name.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**xs:import**

Adds multiple schemas with different target namespace to a document. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-import](http://www.w3.org/TR/xmlschema11-1/#element-import).

**xs:import properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Location</td>
<td>Imported schema location</td>
<td>Any URI</td>
</tr>
<tr>
<td>Namespace</td>
<td>Imported schema namespace</td>
<td>Any URI</td>
</tr>
<tr>
<td>ID</td>
<td>Import ID</td>
<td>Any ID</td>
</tr>
</tbody>
</table>
### xs:redefine

Redefines simple and complex types, groups, and attribute groups from an external schema. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-redefine](http://www.w3.org/TR/xmlschema11-1/#element-redefine).

#### xs:redefine properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Location</td>
<td>Redefine schema location.</td>
<td>Any URI</td>
</tr>
<tr>
<td>ID</td>
<td>Redefine ID</td>
<td>Any ID</td>
</tr>
<tr>
<td>Component</td>
<td>The component name.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

### xs:override

The override construct allows replacements of old components with new ones without any constraint. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-override](http://www.w3.org/TR/xmlschema11-1/#element-override).

#### xs:override properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Location</td>
<td>Redefine schema location.</td>
<td>Any URI</td>
</tr>
<tr>
<td>ID</td>
<td>Redefine ID</td>
<td>Any ID</td>
</tr>
</tbody>
</table>

### xs:notation

Describes the format of non-XML data within an XML document. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-notation](http://www.w3.org/TR/xmlschema11-1/#element-notation).

#### xs:notation properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible values</th>
<th>Mentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The notation name. Always required.</td>
<td>Any NCName.</td>
<td>If missing, will be displayed as '[notation]' in diagram.</td>
</tr>
<tr>
<td>System Identifier</td>
<td>The notation system identifier.</td>
<td>Any URI</td>
<td>Required if public identifier is absent, otherwise optional.</td>
</tr>
<tr>
<td>Public Identifier</td>
<td>The notation public identifier.</td>
<td>A Public ID value</td>
<td>Required if system identifier is absent, otherwise optional.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID</td>
<td>For all notations.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
<td>For all notations.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property.</td>
<td>For all notations.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
<td>For all notations.</td>
</tr>
</tbody>
</table>
**xs:sequence, xs:choice, xs:all**

**Figure 83: An xs:sequence in diagram**

`xs:sequence` specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-sequence](http://www.w3.org/TR/xmlschema11-1/#element-sequence).

**Figure 84: An xs:choice in diagram**

`xs:choice` allows only one of the elements contained in the declaration to be present within the containing element. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-choice](http://www.w3.org/TR/xmlschema11-1/#element-choice).

**Figure 85: An xs:all in diagram**

`xs:all` specifies that the child elements can appear in any order. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-all](http://www.w3.org/TR/xmlschema11-1/#element-all).

The compositor graphical representation also contains the value for the `minOccurs` and `maxOccurs` properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the compositor are drawn using dotted lines if the compositor is optional.

**xs:sequence, xs:choice, xs:all properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compositor</td>
<td>Compositor type.</td>
<td>sequence, choice, all.</td>
<td>'all' is only available as a child of a group or complex type.</td>
</tr>
<tr>
<td>Min Occurs</td>
<td>Minimum occurrences of compositor.</td>
<td>A numeric positive value. Default is 1.</td>
<td>The property is not present if compositor is 'all' and is child of a group.</td>
</tr>
<tr>
<td>Max Occurs</td>
<td>Maximum occurrences of compositor.</td>
<td>A numeric positive value. Default is 1.</td>
<td>The property is not present if compositor is 'all' and is child of a group.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID</td>
<td>For all compositors.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
<td>For all compositors.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
<td>For all compositors.</td>
</tr>
</tbody>
</table>

**xs:any**

Enables the author to extend the XML document with elements not specified by the schema. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-any](http://www.w3.org/TR/xmlschema11-1/#element-any).

The graphical representation also contains the value for the `minOccurs` and `maxOccurs` properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the wildcard are drawn using dotted lines if the wildcard is optional.

**xs:any properties**
### Possible Values

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td>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).</td>
<td>##any, ##other, ##targetNamespace, ##local or anyURI</td>
</tr>
<tr>
<td>notNamespace</td>
<td>Specifies the namespace that extension elements or attributes cannot come from.</td>
<td>##local, ##targetNamespace</td>
</tr>
<tr>
<td>notQName</td>
<td>Specifies an element or attribute that is not allowed.</td>
<td>##defined</td>
</tr>
<tr>
<td>Process Contents</td>
<td>Type of validation required on the elements allowed for this wildcard.</td>
<td>skip, lax, strict</td>
</tr>
<tr>
<td>Min Occurs</td>
<td>Minimum occurrences of any</td>
<td>A numeric positive value. Default is 1.</td>
</tr>
<tr>
<td>Max Occurs</td>
<td>Maximum occurrences of any</td>
<td>A numeric positive value. Default is 1.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The name of the edited component.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

### xs:anyAttribute

Enables the author to extend the XML document with attributes not specified by the schema. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-anyAttribute](http://www.w3.org/TR/xmlschema11-1/#element-anyAttribute).

### xs:anyAttribute properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td>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).</td>
<td>##any, ##other, ##targetNamespace, ##local or anyURI</td>
</tr>
<tr>
<td>Process Contents</td>
<td>Type of validation required on the elements allowed for this wildcard.</td>
<td>skip, lax, strict</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The name of the edited component.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>
**xs:unique**

Defines that an element or an attribute value must be unique within the scope. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-unique](http://www.w3.org/TR/xmlschema11-1/#element-unique).

**xs:unique properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The unique name. Always required.</td>
<td>Any NCName.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**xs:key**

Specifies an attribute or element value as a key (unique, non-nullable and always present) within the containing element in an instance document. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-key](http://www.w3.org/TR/xmlschema11-1/#element-key).

**xs:key properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The key name. Always required.</td>
<td>Any NCName.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**xs:keyRef**

Specifies that an attribute or element value corresponds to that of the specified key or unique element. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-keyref](http://www.w3.org/TR/xmlschema11-1/#element-keyref).
A keyref by default displays the *Referenced Key* property when rendered.

### xs:keyRef properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The keyref name. Always required.</td>
<td>Any NCName.</td>
</tr>
<tr>
<td>Referenced Key</td>
<td>The name of referenced key.</td>
<td>any declared element constraints.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The component namespace.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**xs:selector**

Specifies an XPath expression that selects a set of elements for an identity constraint. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-selector](http://www.w3.org/TR/xmlschema11-1/#element-selector).

### xs:selector properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPath</td>
<td>Relative XPath expression identifying the element on which the constraint applies.</td>
<td>An XPath expression.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**xs:field**

Specifies an XPath expression that specifies the value used to define an identity constraint. See more info at [http://www.w3.org/TR/xmlschema11-1/#element-field](http://www.w3.org/TR/xmlschema11-1/#element-field).

### xs:field properties

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

**xs:assert**

Assertions provide a flexible way to control the occurrence and values of elements and attributes available in an XML Schema.

*Note: xs:assert is available for XML Schema 1.1.*
xs:assert properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>Specifies an XPath expression. If the XPath condition is valid, the specified type is selected as the element type. The expressions allowed are limited to a subset of XPath 2.0. Only the attributes of the current element and inheritable attributes from ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.</td>
<td>An XPath expression.</td>
</tr>
<tr>
<td>XPath Default Namespace</td>
<td>The default namespace used when the XPath expression is evaluated.</td>
<td>##defaultNamespace,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>##targetNamespace,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>##local.</td>
</tr>
<tr>
<td>ID</td>
<td>Specifies the component ID.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**xs:openContent**

The `openContent` element enables instance documents to contain extension elements interleaved among the elements declared by the schema. You can declare open content for your elements at one place - within the complexType definition, or at the schema level.

For further details about the `openContent` component, go to [http://www.w3.org/TR/xmlschema11-1/#element-openContent](http://www.w3.org/TR/xmlschema11-1/#element-openContent).

**xs:openContent properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Specifies where the extension elements can be inserted.</td>
<td>The value can be: &quot;interleave&quot;, &quot;suffix&quot; or &quot;none&quot;. The default value is &quot;interleave&quot;.</td>
</tr>
<tr>
<td>ID</td>
<td>The component id.</td>
<td>Any ID.</td>
</tr>
<tr>
<td>Component</td>
<td>The edited component name.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

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

**Constructs Used to Group Schema Components**

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

- Attributes
- Constraints
- Substitutions
**Attributes**

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

**Attributes properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>The element for which the attributes are displayed.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**Constraints**

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

**Attributes properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>The element for which the constraints are displayed.</td>
<td>Not editable property.</td>
</tr>
<tr>
<td>System ID</td>
<td>The component system id.</td>
<td>Not editable property.</td>
</tr>
</tbody>
</table>

**Substitutions**

Groups all elements which can substitute the current element.

**Attributes properties**
### Navigation in the Schema Diagram

The following editing and navigation features work for all types of schema components:

- Select consecutive components on the diagram (components from the same level) using the Shift key. You can also make discontinuous selections in the schema diagram using the Ctrl (Meta on Mac OS) key. To deselect one of the components, use Ctrl Click (Command Click on OS X).
- Use the arrow keys to navigate the diagram vertically and horizontally.
- Use Home/End keys to navigate to the first/last component from the same level. Use Ctrl Home (Command Home on OS X) key combination to go to the diagram root and Ctrl End (Command End on OS X) to go to the last child of the selected component.
- You can easily go back to a previously visited component while moving from left to right. The path will be preserved only if you use the left arrow key or right arrow key. For example, if the current selection is on the second attribute from an attribute group and you press the left arrow key to navigate to the attribute group, when you press the right arrow key, then the selection will be moved to the second attribute.
- Go back and forward between components viewed or edited in the diagram by selecting them in the Outline view:
  - Back (go to previous schema component).
  - Forward (go to next schema component).
  - Go to Last Modification (go to last modified schema component).
- Copy, reference, or move global components, attributes, and identity constraints to a different position and from one schema to another using the Cut/Copy and Paste/Paste as Reference actions.
- Go to the definition of an element or attribute with the Show Definition action.
- You can expand and see the contents of the imports/includes/redefines in the diagram. In order to edit components from other schemas the schema for each component will be opened as a separate file in Oxygen XML Developer plugin.

**Tip:** If an XML Schema referenced by the current opened schema was modified on disk, the change will be detected and you will be asked to refresh the current schema contents.

- Recursive references are marked with a recurse symbol: 
  Click this symbol to navigate between the element declaration and its reference.

---

### Schema Validation

Validation for the Design mode is seamlessly integrated in the Oxygen XML Developer plugin XML documents validation capability.
A schema validation error is presented by highlighting the invalid component:

- In the **Attributes View**.
- In the diagram by surrounding the component that has the error with a red border.

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

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

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

If you validate the entire schema using the Validate action from the XML menu or from the Validation toolbar drop-down list, all validation errors will be presented in the Errors tab. To resolve an error, just click on it (or double click for errors located in other schemas) and the corresponding schema component will be displayed as the diagram root so that you can easily correct the error.

**Important:** If the schema imports only the namespace of other schema without specifying the schema location and a catalog is set-up that maps the namespace to a certain location both the validation and the diagram will correctly identify the imported schema.

**Tip:** If the validation action finds that the schema contains unresolved references, the application will suggest the use of validation scenarios, but only if the current edited schema is a XML Schema module.

### Schema Editing Actions

You can edit an XML schema using drag and drop operations or contextual menu actions.

Drag and drop is the easiest way to move the existing components to other locations in an XML schema. For example, you can quickly insert an element reference in the diagram with a drag and drop from the Outline view to a compositor in the diagram. Also, the components order in an xs:sequence can be easily changed using drag and drop.

If this property has not been set, you can easily set the attribute/element type by dragging over it a simple type or complex type from the diagram. If the type property for a simple type or complex type is not already set, you can set it by dragging over it a simple or complex type.
Depending on the drop area, different actions are available:

- **move** - Context dependent, the selected component is moved to the destination.
- **reference** - Context dependent, the selected component is referenced from the parent.
- **copy** - If \(\text{Ctrl (Meta on Mac OS)}\) key is pressed, a copy of the selected component is inserted to the destination.

Visual clues about the operation type are indicated by the mouse pointer shape:

- ![arrow] - When moving a component.
- ![pointer] - When referencing a component.
- ![copy] - When copying a component.

You can edit some schema components directly in the diagram. For these components, you can edit the name and the additional properties presented in the diagram by double clicking the value you want to edit. If you want to edit the name of a selected component, you can also press **(Enter)**. The list of properties which can be displayed for each component can be customized in the **Preferences**.

When editing references, you can choose from a list of available components. Components from an imported schema for which the target namespace does not have an associated prefix is displayed in the list as `componentName#targetNamespace`. If the reference is from a target namespace which was not yet mapped, you are prompted to add prefix mappings for the inserted component namespace in the current edited schema.

You can also change the compositor by double-clicking it and choose the compositor you want from the proposals list.

There are some components that cannot be edited directly in the diagram: imports, includes, redefines. The editing action can be performed if you double-click or press **(Enter)** on an import/include/redefine component. An edit dialog is displayed, allowing you to customize the directives.

### Contextual Menu Actions in the Design Mode

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

- **Show Definition (\text{Ctrl Shift ENTER (Command Shift ENTER on OS X)})**
  Shows the definition for the current selected component. For references, this action is available by clicking the arrow displayed in its bottom right corner.

- **Open Schema (\text{Ctrl Shift ENTER (Command Shift ENTER on OS X)})**
  Opens the selected schema. This action is available for `xsd:import`, `xsd:include` and `xsd:redefine` elements. If the file you try to open does not exist, a warning message is displayed and you have the possibility to create the file.

- **Edit Attributes... (\text{Alt Shift Enter})**
  Allows you to edit the attributes of the selected component in a dialog box that presents the same attributes as in the **Attributes View** and the **Facets View**. The actions that can be performed on attributes in this dialog box are the same actions presented in the two views.

- **Append child**
  Offers a list of valid components to append depending on the context. For example to a complex type you can append a compositor, a group, attributes or identity constraints (\text{unique}, \text{key}, \text{keyref}). You can set a name for a named component after it was added in the diagram.

- **Insert before**
  Inserts before the selected component in the schema. The list of components that can be inserted depends on the context. For example, before an `xsd:import` you can insert an `xsd:import`, `xsd:include` or `xsd:redefine`. You can set a name for a named component after it was added in the diagram.
Insert after
Inserts a component after the selected component on the schema. The list of components that can be inserted depends on the context. You can set a name for a named component after it was added in the diagram.

New global
Inserts a global component in the schema diagram. This action does not depend on the current context. If you choose to insert an import you have to specify the URL of the imported file, the target namespace and the import ID. The same information, excluding the target namespace, is requested for an xsd:include or xsd:redefine element.

Note: If the imported file has declared a target namespace, the field Namespace is completed automatically.

Edit Schema Namespaces...
When performed on the schema root, it allows you to edit the schema target namespace and namespace mappings. You can also invoke the action by double-clicking the target namespace property from Attributes view for the schema or by double-clicking the schema component.

Edit Annotations...
Allows you to edit the annotation for the selected schema component in the Edit Annotations dialog box. You can perform the following operations in the dialog box:

- **Edit all appinfo/documentation items for a specific annotation** - all appinfo/documentation items for a specific annotation are presented in a table and can be easily edited. Information about an annotation item includes: type (documentation/appinfo), content, source (optional, specify the source of the documentation/appinfo element) and xml:lang. The content of a documentation/appinfo item can be edited in the Content area below the table.

  - **Insert/Insert before/Remove documentation/appinfo.** The Add button allows you to insert a new annotation item (documentation/appinfo). You can add a new item before the item selected in table by pressing the Insert Before button. Also, you can delete the selected item using the Remove button.

  - **Move items up/down** - to do this use the Move up and Move down buttons.

  - **Insert/Insert before/Remove annotation** - available for components that allow multiple annotations like schemas or redefines.

  - **Specify an ID for the component annotation.** the ID is optional.

Annotations are rendered by default under the graphical representation of the component. When you have a reference to a component with annotations, these annotations are presented in the diagram also below the reference component. The Edit Annotations action invoked from the contextual menu edit the annotations for the reference. If the reference component does not have annotations, you can edit the annotations of the referenced component by double-clicking the annotations area. Otherwise you can edit the referenced component annotations only if you go to the definition of the component.

Note: For imported/included components which do not belong to the currently edited schema, the Edit Annotations dialog box presents the annotation as read-only. To edit its annotation, open the schema where the component is defined.

Extract Global Element
Action available for local elements. A local element is made global and is replaced with a reference to the global element. The local element properties that are also valid for the global element declaration are kept.

![Figure 87: Extracting a Global Element](image)
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 88: Extracting a Global Attribute**

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

Action available for compositors (sequence, choice, all). This action extracts a global group and makes a reference to it. The action is enabled only if the parent of the compositor is not a group.

If you execute Extract Global Group on the sequence element, the Extract Global Component dialog box is shown and you can choose a name for the group. If you type personGroup, the result is:

Figure 89: 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.
If you use the action on the `union` component and choose `numericST` for the new global simple type name, the result is:

![Diagram showing the result of extracting a global simple type](image)

**Figure 90: Extracting a Global Simple Type**

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

![Diagram showing the result of extracting a global complex type](image)

**Figure 91: Extracting a Global Complex Type**

**Rename Component in...**

Rename the selected component.
Cut Ctrl X (Command X on OS X)
Cut the selected component(s).

Copy Ctrl C (Command C on OS X)
Copy the selected component(s).

Copy XPath
This action copies an XPath expression that identifies the selected element or attribute in an instance XML document of the edited schema and places it in the clipboard.

Paste Ctrl V (Command V on OS X)
Paste the component(s) from the clipboard as children of the selected component.

Paste as Reference
Create references to the copied component(s). If not possible a warning message is displayed.

Remove (Delete)
Remove the selected component(s).

Override component
Copies the overridden component in the current XML Schema. This option is available for \texttt{xs:override} components.

Redefine component
The referenced component is added in the current XML Schema. This option is available for \texttt{xs:redefine} components.

Optional
Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The \texttt{minOccurs} property is set to 0 and the \texttt{use} property for attributes is set to \texttt{optional}.

Unbounded
Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The \texttt{maxOccurs} property is set to \texttt{unbounded} and the \texttt{use} property for attributes is set to \texttt{required}.

Search
Can be performed on local elements or attributes. This action makes a reference to a global element or attribute.

Search References
Searches all references of the item found at current cursor position in the defined scope if any.

Search References in...
Searches all references of the item found at current cursor position in the specified scope.

Search Occurrences in File
Searches all occurrences of the item found at current cursor position in the current file.

Component Dependencies
Allows you to see the dependencies for the current selected component.

Resource Hierarchy
Allows you to see the hierarchy for the current selected resource.

Flatten Schema
Recursively adds the components of included Schema files to the main one. It also flattens every imported XML Schema from the hierarchy.

Resource Dependencies
Allows you to see the dependencies for the current selected resource.

Expand all
Expands recursively all sub-components of the selected component.
Collapse all
Collapses recursively all sub-components of the selected component.

Save as Image...
Save the diagram as image, in JPEG, BMP, SVG or PNG format.

Generate Sample XML Files
Generate XML files using the current opened schema. The selected component is the XML document root. See more in the Generate Sample XML Files section.

Options...
Show the Schema preferences panel.

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

![Outline View for XML Schema](image)

**Figure 92:** The Outline View for XML Schema

The Outline view provides the following options:

- **Selection update on caret move**
  Allows a synchronization between Outline view and schema diagram. The selected view from the diagram is also selected in the Outline view.

- **Sort**
  Allows you to sort alphabetically the schema components.
Show all components
Displays all components that were collected starting from the main files. Components that are not referable from the current file are marked with an orange underline. To reference them, add an import directive with the componentNS namespace.

Show referable components
Displays all components (collected starting from the main files) that can be referenced from the current file. This option is set by default.

Show only local components
Displays the components defined in the current file only.

Group by location/namespace/type
These three operations allow you to group the components by location, namespace, or type. When grouping by namespace, the main schema target namespace is the first presented in the Outline view.

The following contextual menu actions are available:

Remove (Delete)
Removes the selected item from the diagram.

Search References
Searches all references of the item found at current cursor position in the defined scope, if any.

Search References in...
Searches all references of the item found at current cursor position in the specified scope.

Component Dependencies
Allows you to see the dependencies for the current selected component.

Resource Hierarchy
Allows you to see the hierarchy for the current selected resource.

Resource Dependencies
Allows you to see the dependencies for the current selected resource.

Rename Component in...
Renames the selected component.

Generate Sample XML Files...
Generate XML files using the current opened schema. The selected component is the XML document root.

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Tip: The search filter is case insensitive. The following wildcards are accepted:
- * - any string
- ? - any character
- , - patterns separator

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

The Outline content and the editing area are synchronized. When you select a component in the Outline view, its definition is highlighted in the editing area.

The Attributes View
The Attributes view presents the properties for the selected component in the schema diagram. If hidden, you can open it from Window > Show View > Other > oXygen > Attributes.
Figure 93: The Attributes View

The default value of a property is presented in the Attributes view with blue foreground. The properties that can't be edited are rendered with gray foreground. A non-editable category which contains at least one child is rendered with bold. Bold properties are properties with values set explicitly to them.

Properties for components which do not belong to the current edited schema are read-only but if you double-click them you can choose to open the corresponding schema and edit them.

You can edit a property by double-clicking on by pressing Enter. For most properties you can choose valid values from a list or you can specify another value. If a property has an invalid value or a warning, it will be highlighted in the table with the corresponding foreground color. By default properties with errors are highlighted with red and the properties with warnings are highlighted with yellow. You can customize these colors from the Document checking user preferences.

For imports, includes and redefines, the properties are not edited directly in the Attributes view. A dialog will be shown allowing you to specify properties for them.

The schema namespace mappings are not presented in Attributes view. You can view/edit these by choosing Edit Schema Namespaces from the contextual menu on the schema root. See more in the Edit Schema Namespaces section.

The Attributes view has five actions available on the toolbar and also on the contextual menu:

- **Add**
  Allows you to add a new member type to an union's member types category.

- **Remove**
  Allows you to remove the value of a property.

- **Move Up**
  Allows you to move up the current member to an union's member types category.

- **Move Down**
  Allows you to move down the current member to an union's member types category.

- **Copy**
  Copy the attribute value.

- **Show Definition** Ctrl (Meta on MAC OS) + Click
  Shows the definition for the selected type.
Show Facets

Allows you to edit the facets for a simple type.

The Facets View

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

![Facets View](image)

Figure 94: The Facets View

The default value of a facet is rendered in the Facets view with a blue color. The facets that can't be edited are rendered with a gray color. The grouping categories (for example: Enumerations and Patterns) are not editable. If these categories contain at least one child they are rendered with bold. Bold facets are facets with values set explicitly to them.

⚠️ **Important:** Usually inherited facets are presented as default in the Facets view but if patterns are inherited from a base type and also specified in the current simple type only the current specified patterns will be presented. You can see the effective pattern value obtained by combining the inherited and the specified patterns as a tooltip on the Patterns category.

Facets for components which do not belong to the current edited schema are read-only but if you double-click them you can choose to open the corresponding schema and edit them.

You can edit a facet by double-clicking on it or by pressing Enter, when that facet is selected. For some facets you can choose valid values from a list or you can specify another value. If a facet has an invalid value or a warning, it will be highlighted in the table with the corresponding foreground color. By default facets with errors are presented with red and the facets with warnings with yellow. You can customize the error colors from the Document Checking user preferences.

The Facets view has four toolbar actions available also on the contextual menu:

- **Add**
  - Allows you to add a new enumeration or a new pattern.

- **Remove**
  - Allows you to remove the value of a facet.

- **Move Up**
  - Allows you to move up the current enumeration/pattern in Enumerations/Patterns category.

- **Move Down**
  - Allows you to move down the current enumeration/pattern in Enumerations/Patterns category.

- **Copy**
  - Copy the attribute value.

Open in Regular Expressions Builder

- Allows you to open the pattern in the XML Schema Regular Expressions Builder.
Facets can be fixed to prevent a derivation from modifying its value. To fix a facet value just press the Pin button.

**Editing Patterns**

You are able to edit regular expressions either by hand, or using the Open in Regular Expression Builder action from the contextual menu. This action offers a full-fledged XML Schema Regular Expression builder that guides through testing and constructing the pattern.

**The Palette View**

The Palette view is designed to offer quick access to XML Schema components and to improve the usability of the XML Schema diagram builder. You can use the Palette to drag and drop components in the Design mode. The components offered in the Palette view depend on the XML schema version set in the XML Schema preferences page.

**Figure 95: Palette View**

Components are organized functionally into 4 collapsible categories:

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

**Note:** The type alternative, open content, override, and assert components are available for XML Schema 1.1.

To add a component to the edited schema:

- Click and hold a graphic symbol from the Palette view, then drag the component into the Design view.
- A line dynamically connects the component with the XML schema structure.
- Release the component into a valid position.

**Note:** You cannot drop a component into an invalid position. When you hover the component into an invalid position, the mouse cursor changes its shape into Also, the connector line changes its color from the usual dark grey to the color defined in the Validation error highlight color option (default color is red).

**Edit Schema Namespaces**

You can use the dialog **XML Schema Namespaces** to easily set a target namespace and define namespace mappings for a newly created XML Schema. In the **Design** mode these namespaces can be modified anytime by choosing **Edit Schema Namespaces** from the contextual menu. Also you can do that by double-clicking on the schema root in the diagram.

The **XML Schema Namespaces** dialog allows you to edit the following information:

- **Target namespace** - The target namespace of the schema.
- **Prefixes** - The dialog shows a table with namespaces and the mapped prefixes. You can add a new prefix mapping or remove an already existing one.

**XML Schema Text Editing Mode**

This page is used for editing the XML Schema in a text mode. It offers powerful content completion support, a synchronized Outline view, and multiple refactoring actions. The Outline view has two display modes: the **standard outline** mode and the **components** mode.

A diagram of the XML Schema can be presented side by side with the text. To activate the diagram presentation, enable the **Show Full Model XML Schema diagram** check box from the **Diagram** preferences page.

**Content Completion**

The intelligent **Content Completion Assistant** available in Oxygen XML Developer plugin enables rapid, in-line identification and insertion of elements, attributes and attribute values valid in the editing context. All available proposals are listed in a pop-up list displayed at the current caret position.

The **Content Completion Assistant** is enabled by default. To disable it, open the **Preferences** dialog box and go to **Editor > Content Completion**. It is activated:

- automatically, after a configurable delay from the last key press of the < character. You can adjust the delay from the **Content Completion preferences page**
- on demand, by pressing **Ctrl Space (Command Space on OS X)** on a partial element or attribute name.

**Note:** If the Content Completion list contains only one valid proposal, when you press the **Ctrl Space (Command Space on OS X)** shortcut key, the proposal is automatically inserted.

When active, it displays a list of context-sensitive proposals valid at the current caret position. Elements are highlighted in the list using the Up and Down cursor keys on your keyboard. For each selected item in the list, the **Content Completion Assistant** displays a documentation window. You can customize the size of the documentation window by dragging its top, right, and bottom borders.

To insert the selected content:

- press Enter or Tab on your keyboard to insert both the start and end tags.
- press **Ctrl Enter (Command Enter on OS X)** on your keyboard. Oxygen XML Developer plugin inserts both the start and end tags and positions the cursor between the tags, so you can start typing content.

Depending on the **selected schema version**, Oxygen XML Developer plugin populates the proposals list with information taken either from XML Schema 1.0 or 1.1.

Oxygen XML Developer plugin helps you to easily reference a component by providing the list of proposals (complex types, simple types, elements, attributes, groups, attribute groups, or notations) valid in the current context. The components are collected from the current file or from the imported/included schemas.

When editing **xs:annotation/xs:appinfo** elements of an XML Schema, the **Content Completion Assistant** proposes elements and attributes from a custom schema (by default ISO Schematron). This feature can be configured from the **XSD Content Completion** preferences page.
Highlight Component Occurrences

When a component (for example types, elements, attributes) is found at current cursor position, Oxygen XML Developer plugin performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document. Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is on by default. To configure it, open the Preferences dialog box and go to Editor > Mark Occurrences. A search can also be triggered with the Search > Search Occurrences in File () contextual menu action. All matches are displayed in a separate tab of the Results view.

Editing XML Schema in the Master Files Context

Smaller interrelated modules that define a complex XML Schema cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, an element defined in a main schema document is not visible when you edit an included module. Oxygen XML Developer plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You can set a main XML document either using the master files support from the Navigator view, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Developer plugin warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

• correct validation of a module in the context of a larger schema structure;
• Content Completion Assistant displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one;
• the Outline displays the components collected from the entire schema structure;

Searching and Refactoring Actions in XML Schemas

Search Actions

The following search actions can be applied on attribute, attributeGroup, element, group, key, unique, keyref, notation, simple, or complex types and are available from the Search submenu in the contextual menu of the current editor:

• Search References - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
• Search References in... - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the Search References dialog box.
• Search Declarations - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
• Search Declarations in... - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
• Search Occurrences in File - Searches all occurrences of the item at the caret position in the currently edited file.
Refactoring Actions

The following refactoring actions can be applied on attribute, attributeGroup, element, group, key, unique, keyref, notation, simple, or complex types and are available from the Refactoring submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.

- **Rename Component in...** - Opens the Rename component_type dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the Preview button, you can view the files affected by the Rename Component action.

![Figure 96: Rename Identity Constraint Dialog Box](image)

Component Dependencies View

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

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

- Select the desired schema component in the editor.
- Choose the Component Dependencies action from the contextual menu.

The action is available for all named components (for example elements or attributes).
In the Component Dependencies view the following actions are available in the toolbar:

- **Refresh**
  Refreshes the dependencies structure.

- **Stop**
  Stops the dependencies computing.

- **Configure**
  Allows you to configure a search scope to compute the dependencies structure.

- **History**
  Contains a list of previously executed dependencies computations.

The contextual menu contains the following actions:

- **Go to First Reference**
  Selects the first reference of the referenced component from the current selected component in the dependencies tree.

- **Go to Component**
  Shows the definition of the currently selected component in the dependencies tree.

**Tip:** If a component contains multiple references to another components, a small table is shown containing all these references. When a recursive reference is encountered, it is marked with a special icon ✢.
XML Schema Quick Assist Support

The Quick Fix support improves the development work flow, offering fast access to the most commonly used actions when you edit XML Schema documents.

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

![Quick Assist Support](image)

**Figure 98: Quick Assist Support**

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

- **Rename Component in...**
  Renames the component and all its dependencies.

- **Search Declarations**
  Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

- **Search References**
  Searches all references of the component in a predefined scope.

- **Component Dependencies**
  Searches the component dependencies in a predefined scope.

- **Change Scope...**
  Configures the scope that will be used for future search or refactor operations.

- **Rename Component**
  Allows you to rename the current component in-place.

- **Search Occurrences**
  Searches all occurrences of the component within the current file.

To watch our video demonstration about improving schema development using the **Quick Assist** action set, go to [http://oxygenxml.com/demo/Quick_Assist.html](http://oxygenxml.com/demo/Quick_Assist.html).

XML Schema Resource Hierarchy / Dependencies View

The **Resource Hierarchy / Dependencies** view allows you to easily see the hierarchy / dependencies for an XML Schema, a Relax NG schema or an XSLT stylesheet. To open this view, go to Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies.

The **Resource Hierarchy / Dependencies** is useful when you want to start from an XML Schema (XSD) file and build and review the hierarchy of all the other XSD files that are imported, included or redefined in the given XSD file. The view is also able to build the tree structure, that is the structure of all other XSD files that import, include or redefine the given XSD file. The scope of the search is configurable: the current project, a set of local folders, etc.
The build process for the hierarchy view is started with the **Resource Hierarchy** action available on the contextual menu of the editor panel.

![Hierarchy View](image1)

**Figure 99: Resource Hierarchy/Dependencies View - Hierarchy for xhtml11.xsd**

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

![Dependencies View](image2)

**Figure 100: Resource Hierarchy/Dependencies View - Dependencies for xhtml-param-1.xsd**

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

- **Refresh**
  - Refreshes the Hierarchy/Dependencies structure.
Stop
Stops the hierarchy/dependencies computing.

Show Hierarchy
Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies
Allows you to choose a resource to compute the dependencies structure.

Configure
Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History
Provides access to the list of previously computed dependencies. Use the Clear history button to remove all items from this list.

The contextual menu contains the following actions:

Open
Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location
Copies the location of the resource.

Move resource
Moves the selected resource.

Rename resource
Renames the selected resource.

Show Resource Hierarchy
Shows the hierarchy for the selected resource.

Show Resource Dependencies
Shows the dependencies for the selected resource.

Add to Master Files
Adds the currently selected resource in the Master Files directory.

Expand All
Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All
Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

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

Note: The Move resource or Rename resource actions give you the option to update the references to the resource.

Moving/Renaming XML Schema Resources
You are able to move and rename a resource presented in the Resource/Hierarchy Dependencies view, using the Rename resource and Move resource refactoring actions from the contextual menu.

When you select the Rename action in the contextual menu of the Resource/Hierarchy Dependencies view, the Rename resource dialog box is displayed. The following fields are available:

- New name - Presents the current name of the edited resource and allows you to modify it.
- Update references - Enable this option to update the references to the resource you are renaming.
When you select the Move action from the contextual menu of the Resource/Hierarchy Dependencies view, the Move resource dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

**Generating Documentation for an XML Schema**

Oxygen XML Developer plugin can generate detailed documentation for the components of an XML Schema in HTML, PDF and DocBook XML formats. You can select the components and the level of detail. The components are hyperlinked in both HTML and DocBook documents.

**Note:** You can generate documentation both for XML Schema version 1.0 and 1.1.

To generate documentation for an XML Schema document, select XML Schema Documentation... from the XML Tools > Generate Documentation menu or from the Generate XML Schema Documentation action from the contextual menu of the Navigator view.

![Image of the XML Schema Documentation dialog box]

**Figure 101: The Output panel of the XML Schema Documentation Dialog Box**

The Schema URL field of the dialog box must contain the full path to the XML Schema (XSD) file you want to generate documentation for. The schema may be a local or a remote one. You can specify the path to the schema using the editor variables.

The following options are available in the Settings tab:
• **Format** - Allows you to choose between three predefined formats (*HTML*, *PDF*, *DocBook*) and a custom one (*Custom*). This allows you to control the output format by 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/System Application** - Opens the result in the system application associated with the output file type.

  **Note:** To set the browser or system application that will be used, *open the Preferences dialog box*, then go to *General > Web Browser*. This will take precedence over the default system application settings.

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

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

---

**Figure 102: The Settings Panel of the XML Schema Documentation Dialog Box**

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

• **Diagram** - Displays the diagram for each component. You can choose the image format (JPEG, PNG, SVG) to use for the diagram section.

• **Diagram annotations** - This option controls whether the annotations of the components presented in the diagram sections are included.

• **Namespace** - Displays the namespace for each component.

• **Location** - Displays the schema location for each component.

• **Type** - Displays the component type if it is not an anonymous one.
• **Type hierarchy** - Displays the types hierarchy.

• **Model** - Displays the model (sequence, choice, all) presented in BNF form. Different separator characters are used depending on the information item used:
  - `xs:all` - its children will be separated by space characters.
  - `xs:sequence` - its children will be separated by comma characters.
  - `xs:choice` - its children will be separated by `|` characters.

• **Children** - Displays the list of component's children.

• **Instance** - Displays an XML instance generated based on each schema element.

• **Used by** - Displays the list of all the components that reference the current one. The list is sorted by component type and name.

• **Properties** - Displays some of the component's properties.

• **Facets** - Displays the facets for each simple type

• **Identity constraints** - Displays the identity constraints for each element. For each constraint there are presented the name, type (unique, key, keyref), reference attribute, selector and field(s).

• **Attributes** - Displays the attributes for the component. For each attribute there are presented the name, type, fixed or default value, usage and annotation.

• **Asserts** - Displays the `assert` elements defined in a complex type. The test, XPath default namespace, and annotation are presented for each assert.

• **Annotations** - Displays the annotations for the component. If you choose **Escape XML Content**, the XML tags are present in the annotations.

• **Source** - Displays the text schema source for each component.

• **Generate index** - Displays an index with the components included in the documentation.

• **Include local elements and attributes** - If checked, local elements and attributes are included in the documentation index.

• **Include resource hierarchy** - specifies whether the resource hierarchy for an XML Schema documentation is generated.

• **Export settings / Load settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

These options are persistent between sessions.

**Generate Documentation in HTML Format**

The HTML documentation contains images corresponding to the schema definitions as the ones displayed by the schema diagram editor. These images are divided in clickable areas which are linked to the definitions of the clicked names of types or elements. The documentation of a definition includes a **Used By** section with links to the other definitions that reference it. If the **Escape XML Content** option is unchecked, the HTML or XHTML tags used inside the `xs:documentation` elements of the input XML Schema for formatting the documentation text (for example `<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 XML Developer plugin's XSD editor panel**.
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.

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 105: 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 106: Documentation for a Schema Component

If the schema contains imported or included modules, their dependencies tree is generated in the documentation.
Generate Documentation in PDF, DocBook or a Custom Format

XML Schema documentation can be also generated in PDF, DocBook, or a custom format. You can choose the format from the Schema Documentation dialog box. For the PDF and DocBook formats, the option to split the output in multiple files is disabled.

When choosing PDF, the documentation is generated in DocBook format and after that a transformation using the FOP processor is applied to obtain the PDF file. To configure the FOP processor, see the FO Processors preferences page.

If you generate the documentation in DocBook format you can apply a transformation scenario on the output file, for example one of the scenarios proposed by Oxygen XML Developer plugin (DocBook PDF or DocBook HTML) or configure your own scenario for it.

For the custom format, you can specify your stylesheet to transform the intermediary XML generated in the documentation process. You have to write your stylesheet based on the schema xsdDocSchema.xsd from \[OXYGEN_DIR\]/frameworks/schema_documentation. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in \[OXYGEN_DIR\]/frameworks/schema_documentation/xsl.

When using a custom format you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Customizing the Generated PDF

You are able to customize the PDF output of the documentation for an XML schema by running two transformations and customizing the intermediate file. To do this, use the following procedure:

1. Customize the \[OXYGEN_DIR\] /frameworks/schema_documentation/xsl/xsdDocDocbook.xsl stylesheet to include the content that you want to add in the PDF output. Add the content in the XSLT template with the match="schemaDoc" attribute between these two elements:

   ```xml
   <info>
     <pubdate><xsl:value-of select="format-date(current-date(),'[Mn] [D], [Y]', 'en', (), ())"></pubdate>
   </info>
   <xsl:apply-templates select="schemaHierarchy"/>
   ```

   **Note:** The content that you insert here has to be wrapped in DocBook markup. For details about working with DocBook take a look at the video demonstration from this address http://www.oxygenxml.com/demo/DocBook_Editing_in_Author.html.

2. Create an intermediary file that holds the content of your XML Schema documentation.
   a. Go to Tools > Generate Documentation > XML Schema Documentation.
b. Click Custom in the XML Schema Documentation dialog box.

c. Go to Options.

d. In the Custom format options dialog box, enable Copy additional resources to the output folder, enter: [OXYGEN_DIR]/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl in the Custom XSL field and click OK.

e. When you return to the Custom format options dialog box, just press the Generate button which will generate a DocBook XML file with an intermediary form of the Schema documentation.

3. Create the final PDF document.

   a. Go to Document > Transformation > Configure Transformation Scenario and click New.

   b. In the New Scenario dialog box, go to the XSL URL field and choose the [OXYGEN_DIR]/frameworks/docbook/oxygen/xsdDocDocbookCustomizationFO.xsl file.

   c. Go to the FO Processor tab and enable Perform FO Processing and XSLT result as input.

   d. Go to the Output tab and select the directory where you want to store the XML Schema documentation output and click OK.

   e. Click Apply Associated.

Generating Documentation From the Command-Line Interface

You can export the settings of the XML Schema Documentation dialog box to an XML file by pressing the Export settings button. With the exported settings file, you can generate the same documentation from the command-line interface by running the following scripts:

- schemaDocumentation.bat on Windows.
- schemaDocumentation.sh (on OS X / Unix / Linux).

The scripts are located in the Oxygen XML Developer plugin installation folder. The scripts can be integrated in an external batch process launched from the command-line interface.

The script command-line parameter is the relative path to the exported XML settings file. The files which are specified with relative paths in the exported XML settings are made absolute, relative to the folder where the script is ran from.

### XML Configuration File

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

You can organize an XML schema on several levels linked by `xs:include` and `xs:import` statements. In some cases, working on such a schema as on a single file is more convenient.

The Flatten Schema operation allows you to flatten an entire hierarchy of XML schemas. Starting with the main XML schema, Oxygen XML Developer plugin calculates its hierarchy by processing the `xs:include` and `xs:import` statements. This action is available either from the contextual menu of the editor.
For the main schema file and for each imported schema, a new flattened schema is generated in the output folder. These schemas have the same name as the original ones.

**Note:** If necessary, the operation renames the resulted schemas to avoid duplicated file names.

**Note:** You can choose the output folder and the name of each generated schema file.

A flattened XML schema is obtained by recursively adding the components of the included schemas into the main one. This means Oxygen XML Developer plugin replaces the `xs:include`, `xs:redefine`, and `xs:override` elements with the ones coming from the included files.

The following options are available in the Flatten Schema dialog:

- **Open the flattened XML Schema file in editor** - opens the main flattened schema in the editing area after the operation completes
- **Use the XML Catalogs when collecting the referenced XML Schemas** - enables resolving the imported and included schemas through the available XML Catalogs;
  
  **Note:** Changing this option triggers the recalculation of the dependencies graph for the main schema.

- **Process the imported XML Schemas resolved through the XML Catalogs** - specifies whether the imported schemas that were resolved through an XML Catalog are flattened
- **Flatten the imported XML Schema(s)** - specifies whether the imported schemas are flattened.
  
  **Note:** For the schemas skipped by the flatten operation, no files are created in the output folder and the corresponding import statements remain unchanged.

To flatten a schema from the command line, run one of the following scripts that come with Oxygen XML Developer plugin:

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

The command line accepts the following parameters:

• `-in:inputSchemaURL` - the input schema URL
• `-outDir:outputDirectory` - the directory where the flattened schemas should be saved
• `-flattenImports:boolean_value` - controls if the imported XML Schemas should be flattened or not. The default value `true`
• `-useCatalogs:boolean_value` - controls if the references to other XML Schemas should be resolved through the available XML Catalogs. The default value `false`
• `-flattenCatalogResolvedImports:boolean_value` - controls if the imported schemas that were resolved through the XML Catalogs should be flattened or not

**Note:** This option is used only when `-useCatalogs` is set to `true`. The default value is `true`.

• `-verbose` - provides information about the current flatten XML Schema operation
• `--help | -help | --h | -h` - prints the available parameters for the operation

**Command Line Example**

```
flattenSchema -in:http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd -outDir:mySchemas/ flattened/xhtml
-flattenImports:true -useCatalogs:true -flattenCatalogResolvedImports:true -verbose
```

Generate Sample XML Files

Oxygen XML Developer plugin offers support to generate sample XML files both from XML schema 1.0 and XML schema 1.1, depending on the XML schema version set in Preferences.

To generate sample XML files from an XML Schema, use the Generate Sample XML Files... action. This action is also available in the contextual menu of the schema Design mode.

The Generate Sample XML Files dialog contains the following tabs:

• **Schema**;
• **Options**;
• **Advanced**.

To watch our video demonstration about generating sample XML files, go to http://oxygenxml.com/demo/Generate_Sample_XML_Files.html.
The Schema Tab

Figure 108: The Generate Sample XML Files Dialog

Complete the dialog as follows:

- **URL** - Schema location as an URL. A history of the last used URLs is available in the drop-down box.
- **Namespace** - Displays the namespace of the selected schema.
- **Document root** - After the schema is selected, this drop-down box is populated with all root candidates gathered from the schema. Choose the root of the output XML documents.
- **Output folder** - Path to the folder where the generated XML instances will be saved.
- **Filename prefix and Extension** - Generated file names have the following format: prefixN.extension, where N represents an incremental number from 0 up to Number of instances - 1.
- **Number of instances** - The number of XML files to be generated.
- **Open first instance in editor** - When checked, the first generated XML file is opened in editor.
- **Namespaces** - Here you can specify the default namespace as well as the proxies (prefixes) for namespaces.
- **Load settings / Export settings** - The current settings can be saved for further usage with the Export settings button, and reloaded when necessary with the Load settings button.

The Options Tab

The Options tab allows you to set specific options for different namespaces and elements.
• **Namespace / Element table** - Allows you to set a namespace for each element name that appears in an XML document instance. The following prefix-to-namespace associations are available:
  - All elements from all namespaces (<ANY> - <ANY>). This is the default setting and can be customized from the **XML Instances Generator** preferences page.
  - All elements from a specific namespace.
  - A specific element from a specific namespace.

• **Settings**
  - **Generate optional elements** - When checked, all elements are generated, including the optional ones (having the minOccurs attribute set to 0 in the schema).
  - **Generate optional attributes** - When checked, all attributes are generated, including the optional ones (having the use attribute set to optional in the schema.)
  - **Values of elements and attributes** - Controls the content of generated attribute and element values. Several choices are available:
    - **None** - No content is inserted;
    - **Default** - Inserts a default value depending of data type descriptor of the particular element or attribute. The default value can be either the data type name or an incremental name of the attribute or element (according to the global option from the **XML Instances Generator** preferences page). Note that type restrictions are ignored when this option is enabled. For example, if an element is of a type that restricts an xs:string with the xs:maxLength facet in order to allow strings with a maximum length of 3, the XML instance generator tool may generate string element values longer than 3 characters.
    - **Random** - Inserts a random value depending of data type descriptor of the particular element or attribute.
Important:
If all of the following are true, the XML Instances Generator outputs invalid values:
• at least one of the restrictions is a regexp;
• the value generated after applying the regexp does not match the restrictions imposed by one of the facets.
This limitation leads to attributes or elements with values set to Invalid.

• **Preferred number of repetitions** - Allows the user to set the preferred number of repeating elements related with minOccurs and maxOccurs facets defined in XML Schema.
  • If the value set here is between minOccurs and maxOccurs, then that value is used;
  • If the value set here is less than minOccurs, then the minOccurs value is used;
  • If the value set here is greater than maxOccurs, then maxOccurs is used.

• **Maximum recursion level** - If a recursion is found, this option controls the maximum allowed depth of the same element.

• **Choice strategy** - Option used in case of xs:choice or substitutionGroup elements. The possible strategies are:
  • **First** - the first branch of xs:choice or the head element of substitutionGroup is always used;
  • **Random** - a random branch of xs:choice or a substitute element or the head element of a substitutionGroup is used.

• **Generate the other options as comments** - Option to generate the other possible choices or substitutions (for xs:choice and substitutionGroup). These alternatives are generated inside comments groups so you can uncomment and use them later. Use this option with care (for example on a restricted namespace and element) as it may generate large result files.

• **Load settings / Export settings** - The current settings can be saved for further usage with the Export settings button, and reloaded when necessary with the Load settings button.

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

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

### The Advanced Tab

<table>
<thead>
<tr>
<th>Strings and values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use incremental attribute/element names as default</td>
</tr>
<tr>
<td>Maximum length: 30</td>
</tr>
<tr>
<td>Performance</td>
</tr>
<tr>
<td>Discard optional elements after nested level: 6</td>
</tr>
</tbody>
</table>

This tab allows you to set advanced options that controls the output values of elements and attributes.

• **Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an a element the generated values are: a1, a2, a3, etc. If not checked, the value is the name of the type of that element / attribute, for example: string, decimal, etc.

• **Maximum length** - The maximum length of string values generated for elements and attributes.

• **Discard optional elements after nested level** - The optional elements that exceed the specified nested level are discarded. This option is useful for limiting deeply nested element definitions that can quickly result in very large XML documents.
XML Schema Regular Expressions Builder

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

Figure 110: XML Schema Regular Expressions Builder Dialog Box

The dialog box contains the following sections:

- **Regular expressions editor** - allows you to edit the regular expression to be tested and used. Content completion is available and presents a list with all the predefined expressions. It is triggered by pressing Ctrl Space (Command Space on OS X).
- **Error display area** - if the edited regular expression is incorrect, an error message will be displayed here. The message contains the description and the exact location of the error. Also, a click on the quick navigation button (←) highlights the error inside the regular expression.
- **Category** combo box - here you can choose from several categories of predefined expressions. The selected category influences the displayed expressions in the Available expressions table.
- **Available expressions** table - holds the available regular expressions and a short description for each of them. The set of expressions depends on the category selected in the previous combo box. You can add an expression in the Regular expressions editor by double-clicking on the expression row in the table. You will notice that in the case of Character categories and Block names the expressions are also listed in complementary format. For example: \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 box will be disabled if the current document is edited in grid mode.

**Note:** Some regular expressions may block indefinitely the Java Regular Expressions engine. If the execution of the regular expression does not end in about five seconds, the application displays a dialog box that allows you to interrupt the operation.

---

**Create an XML Schema From a Relational Database Table**

To create an XML Schema from the structure of a relational database table use the special wizard available in the **Tools menu**.

**XML Schema 1.1**

Oxygen XML Developer plugin offers full support for XML Schema 1.1, including:

- XML Documents Validation and Content Completion Based on XML Schema 1.1;
- XML Schema 1.1 Validation and Content Completion;
- Editing XML Schema 1.1 files in the Schema Design mode;
- The Flatten Schema action;
- Resource Hierarchy/Dependencies and Refactoring Actions;
- Master Files;
- Generating Documentation for XML Schema 1.1;
- Support for generating XML instances based on XML Schema.

XML Schema 1.1 is a superset of XML Schema 1.0, that offers lots of new powerful capabilities.

---

**Figure 111: XML Schema 1.1**

The significant new features in XSD 1.1 are:

- **Assertions** - support to define assertions against the document content using XPath 2.0 expressions (an idea borrowed from Schematron);
- **Conditional type assignment** - the ability to select the type against which an element is validated based on the values of the attribute of the element;
- **Open content** - content models are able to use the `openContent` element to specify content models with **open content**. These content models allow elements not explicitly mentioned in the content model to appear in the document.
instance. It is as if wildcards were automatically inserted at appropriate points within the content model. A schema
document wide default may be set, which causes all content models to be open unless specified otherwise.

To see the complete list with changes since version 1.0, go to http://www.w3.org/TR/xmlschema11-1/#ch_specs.

XML Schema 1.1 is intended to be mostly compatible with XML Schema 1.0 and to have approximately the same scope.
It also addresses bug fixes and brings improvements that are consistent with the constraints on scope and compatibility.

Note: An XML document conforming to a 1.0 schema can be validated using a 1.1 validator, but an XML
document conforming to a 1.1 schema may not validate using a 1.0 validator.

In case you are constrained to use XML Schema 1.0 (for example if you develop schemas for a server that uses an XML
Schema 1.0 validator which cannot be updated), change the default XML Schema version to 1.0. If you keep the default
XML Schema version set to 1.1, the content completion window presents XML Schema 1.1 elements that you can insert
accidentally in an 1.0 XML Schema. So even if you make a document invalid conforming with XML Schema 1.0, the
validation process does not signal any issues.

To change the default XML Schema version, open the Preferences dialog box and go to XML > XML Parser > XML
Schema.

To watch our video demonstration about the XML Schema 1.1 support, go to

Setting the XML Schema Version

Oxygen XML Developer plugin lets you set the version of the XML Schema you are editing either in the XML Schema
preferences page, or through the versioning attributes. In case you want to use the versioning attributes, set the
minVersion and maxVersion attributes, from the http://www.w3.org/2007/XMLSchema-versioning namespace,
on the schema root element.

Note: The versioning attributes take priority over the XML Schema version defined in the preferences page.

Table 3: Using the minVersion and maxVersion Attributes to Set the XML Schema Version

<table>
<thead>
<tr>
<th>Versioning Attributes</th>
<th>XML Schema Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>minVersion = &quot;1.0&quot; maxVersion = &quot;1.1&quot;</td>
<td>1.0</td>
</tr>
<tr>
<td>minVersion = &quot;1.1&quot;</td>
<td>1.1</td>
</tr>
<tr>
<td>minVersion = &quot;1.0&quot; maxVersion = greater than &quot;1.1&quot;</td>
<td>the XML Schema version defined in the XML Schema preferences page.</td>
</tr>
</tbody>
</table>

To change the XML Schema version of the current document, use the Change XML Schema version action from the
contextual menu. This is available both in the Text mode, and in the Design mode and opens the Change XML Schema
version dialog box. The following options are available:

• XML Schema 1.0 - Inserts the minVersion and maxVersion attributes on the schema element and gives them
  the values "1.0" and "1.1" respectively. Also, the namespace declaration
  (xmlns:vc=http://www.w3.org/2007/XMLSchema-versioning) is inserted automatically in case it does not exist.

• XML Schema 1.1 - Inserts the minVersion attribute on the schema element and gives it the value "1.1". Also,
  removes the maxVersion attribute if it exists and adds the versioning namespace
  (xmlns:vc=http://www.w3.org/2007/XMLSchema-versioning) in case it is not declared.

• Default XML Schema version - Removes the minVersion and maxVersion attributes from the schema
  element. The default schema version, defined in the XML Schema preferences page, is used.
**Note:** The **Change XML Schema version** action is also available in the informative panel presented at the top of the edited XML Schema. In case you close this panel, it will no longer appear until you restore Oxygen XML Developer plugin to its default options.

Oxygen XML Developer plugin automatically uses the version set through the versioning attributes, or the default version in case the versioning attributes are not declared, when proposing content completion elements and validating an XML Schema. Also, the XML instance validation against an XML Schema is aware of the versioning attributes defined in the XML Schema.

When you generate sample XML files from an XML Schema, Oxygen XML Developer plugin takes into account the `minVersion` and `maxVersion` attributes defined in the XML Schema.

## Editing XQuery Documents

This section explains the features of the XQuery editor and how to use them.

### XQuery Outline View

The XQuery document structure is presented in the **XQuery Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to a component by knowing its name. It can be opened from the **Window > Show View > Other > oXygen > Outline** menu action.

![Figure 112: XQuery Outline View](image)

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

- **Selection update on caret move**
  
  Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the caret moves or the changes performed in the XQuery editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

- **Sort**
  
  Allows you to alphabetically sort the XQuery components.

- **Show all components**
  
  Displays all collected components starting from the current file. This option is set by default.
Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the Outline view.

If you know the component name, you can search it in the Outline view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, (Enter), (Tab), (Shift-Tab). To switch from tree structure to the filter text field, you can use (Tab), (Shift-Tab).

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

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

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

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Folding in XQuery Documents

In a large XQuery document, the instructions enclosed in the '{' and '}' characters can be collapsed so that only the needed instructions remain in focus. The same folding features available for XML documents are also available in XQuery documents.

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

Figure 113: Folding in XQuery Documents

There is available the action Go to Matching Bracket Ctrl Shift G (Command Shift G on OS X) on contextual menu of XQuery editor for going to matching character when cursor is located at '{' character or '}’ character. It helps for finding quickly matching character of current folding element.
Formatting and Indenting XQuery Documents

Editing XQuery documents may lead to large chunks of content that is not easily readable by human audience. Also, each developer may have a particular way of writing XQuery code. Oxygen XML Developer plugin assists you in maintaining a consistent code writing style by providing the Format and Indent action.

The Format and Indent action achieves this by performing the following steps:

- manages whitespaces, by collapsing or inserting space characters where needed.
- formats complex expressions on multiple, more readable lines by properly indenting each of them. The amount of whitespaces that form an indent unit is controlled through one of the Indent with tabs and Indent size options from the Format Preferences page.

Note: These operations can be performed only if your XQuery document conforms with W3C XQuery 1.0, XQuery Update Facility 1.0, and XQuery 3.0 specifications. If the Format and Indent operation fails, the document is left unaltered and an error message is presented in the Results view.

The Format and Indent action is available in the Document > Source menu and also on the toolbar.

Generating HTML Documentation for an XQuery Document

To generate HTML documentation for an XQuery document, use the XQuery Documentation dialog box. It is opened with the XQuery Documentation... action that is available from the XML Tools > Generate Documentation menu or from the Generate XQuery Documentation action from the contextual menu of the Navigator view.

The dialog box allows you to configure a set of parameters of the process of generating the HTML documentation.

![XQuery Documentation Dialog Box](image)

Figure 114: The XQuery Documentation Dialog Box

The following options are available:

- **Input** - The Input panel allows the user to specify either the File or the Folder which contains the files for which to generate the documentation. One of the two text fields of the Input panel must contain the full path to the XQuery
file. Extensions for the XQuery files contained in the specified directory can be added as comma-separated values. Default there are offered `xquery`, `xq`, `xqy`.

- **Default function namespace** - Optional URI for the default namespace for the submitted XQuery, only if it exists.
- **Predefined function namespaces** - Optional engine dependent, predefined namespaces that the submitted XQuery refers to. They allow the conversion to generate annotation information to support the presentation component hypertext linking, only if the predefined modules have been loaded into the local xqDoc XML repository.

- **Open in Browser/System Application** - Select this option if you want the result to be opened in the system application associated with that file type.

  **Note:** To set the browser or system application that will be used, open the **Preferences** dialog box, then go to **General > Web Browser**. This will take precedence over the default system application settings.

- **Output** - Allows the user to specify where the generated documentation is saved on disk.

### Editing WSDL Documents

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

Oxygen XML Developer plugin provides a special type of editor dedicated to WSDL documents. The WSDL editor offers support to check whether a WSDL document is valid, a specialized Content Completion Assistant, a component oriented **Outline** view, searching and refactoring operations, and support to generate documentation.

Both WSDL version 1.1 and 2.0 are supported and SOAP versions 1.1 and 1.2. That means that in the location where a SOAP extension can be inserted the **Content Completion Assistant** offers elements from both SOAP 1.1 and SOAP 1.2. Validation of SOAP requests is executed first against a SOAP 1.1 schema and then against a SOAP 1.2 schema. In addition to validation against the XSD schemas, Oxygen XML Developer plugin also checks if the WSDL file conforms with the WSDL specification (available only for WSDL 1.1 and SOAP 1.1).

In the following example you can see how the errors are reported.

![Figure 115: Validating a WSDL file](http://www.oxygenxml.com/demo/Create_New_WSDL.html)

To watch our video demonstration about the WSDL editing support in Oxygen XML Developer plugin, go to [http://www.oxygenxml.com/demo/Create_New_WSDL.html](http://www.oxygenxml.com/demo/Create_New_WSDL.html).

### WSDL Outline View

The WSDL **Outline** view displays the list of all the components (services, bindings, port types and so on) of the currently open WSDL document along with the components of its imports.

In case you use the **Master Files support**, the **Outline** view collects the components of a WSDL document starting from the master files of the current document.

To enable the **Outline** view, go to **Window > Show View > Other > oXygen > Outline**.
Figure 116: The WSDL Outline View

The Outline view can display both the components of the current document and its XML structure, organized in a tree-like fashion. You can switch between the components display mode and the XML structure display mode using the Show XML structure and Show components actions. The following actions are available in the View menu on the Outline view action bar when you work with the components display mode:

Filter returns exact matches
The text filter of the Outline view returns only exact matches.

-selection update on caret move
Controls the synchronization between the Outline view and the current document. The selection in the Outline view can be synchronized with the caret moves or the changes in the WSDL editor. Selecting one of the components from the Outline view also selects the corresponding item in the current document.

Show XML structure
Displays the XML structure of the current document in a tree-like structure.

Sort
Sorts the components in the Outline view alphabetically.

Show all components
Displays all the components that were collected starting from current document or from the main document in case it is defined.

Show referable components
Displays all the components that you can reference from the current document.

Show only local components
Displays the components defined in the current file only.

Group by location
Groups the WSDL components by their location.

Group by type
Groups the WSDL components by their type.

Group by namespace
Groups the WSDL components by their namespace.
Note: By default all the three grouping criteria are active.

When you work with the XML structure display mode the following actions are available:

Show components
Switches the Outline view to the components display mode.

Flat presentation mode of the filtered results
When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions
Show/hide comments and processing instructions in the Outline view.

Show element name
Show/hide element name.

Show text
Show/hide additional text content for the displayed elements.

Show attributes
Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from the Outline preferences panel.

Configure displayed attributes
Displays the XML Structured Outline preferences page.

The following contextual menu actions are available in the Outline view when you use it in the components display mode:

Edit Attributes
Opens a dialog that allows you to edit the attributes of the currently selected component.

Cut
Cuts the currently selected component.

Copy
Copies the currently selected component.

Delete
Deletes the currently selected component.

Search references
Searches for the references of the currently selected component.

Search references in
Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies
Displays the dependencies of the currently selected component.

Resource Hierarchy
Displays the hierarchy for the currently selected resource.

Resource Dependencies
Displays the dependencies of the currently selected resource.

Rename Component in...
Renames the currently selected component in the context of a scope that you define.

The following contextual menu actions are available in the Outline view when you use it in the XML structure display mode:
Append Child
Displays a list of elements that you can insert as children of the current element.

Insert Before
Displays a list of elements that you can insert as siblings of the current element, before the current element.

Insert After
Displays a list of elements that you can insert as siblings of the current element, after the current element.

Toggle Comment
Comments/uncomments the currently selected element.

Search references
Searches for the references of the currently selected component.

Search references in
Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies
Displays the dependencies of the currently selected component.

Rename Component in...
Renames the currently selected component in the context of a scope that you define.

Cut
Cuts the currently selected component.

Copy
Copies the currently selected component.

Delete
Deletes the currently selected component.

Expand more
Expands the structure of a component in the Outline view.

Collapse all
Collapses the structure of all the component in the Outline view.

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

Tip: The search filter is case insensitive. The following wildcards are accepted:
- * - any string
- ? - any character
- , - patterns separator

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

The Outline content and the editing area are synchronized. When you select a component in the Outline view, its definition is highlighted in the editing area.

Content Completion in WSDL Documents
The Content Completion Assistant is a powerful feature that enhances the editing of WSDL documents. It helps you define WSDL components by proposing context-sensitive element names. Another important capability of the Content Completion Assistant is to propose references to the defined components when you edit attribute values. For example, when you edit the type attribute of a binding element, the Content Completion Assistant proposes all the defined port types. Each proposal that the Content Completion Assistant offers is accompanied by a documentation hint.

Note: XML schema specific elements and attributes are offered when the current editing context is the internal XML schema of a WSDL document.
Note: The Content Completion Assistant collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the Master Files support go to Defining Master Files at Project Level.

Namespace prefixes in the scope of the current context are presented at the top of the content completion window to speed up the insertion into the document of prefixed elements.

For the common namespaces, like XML Schema namespace (http://www.w3.org/2001/XMLSchema) or SOAP namespace (http://schemas.xmlsoap.org/wsdl/soap/), Oxygen XML Developer plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

Editing WSDL Documents in the Master Files Context

Smaller interrelated modules that define a complex WSDL structure cannot be correctly edited or validated individually, due to their interdependency with other modules. Oxygen XML Developer plugin provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger WSDL structure.

You can set a main WSDL document either using the master files support from the Navigator view, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main modules. Oxygen XML Developer plugin warns you if the current module is not part of the dependencies graph computed for the main WSDL document. In this case, it considers the current module as the main WSDL document.

The advantages of editing in the context of a master file include:

- correct validation of a module in the context of a larger WSDL structure;
- **Content Completion Assistant** displays all components valid in the current context;
- the Outline displays the components collected from the entire WSDL structure.
Note: When you edit an XML schema document that has a WSDL document set as master, the validation operation is performed over the master WSDL document.

To watch our video demonstration about editing WSDL documents in the master files context, go to http://oxygenxml.com/demo/WSDL_Working_Modules.html.

Searching and Refactoring Operations in WSDL Documents

Search Actions

The following search actions are available from the Search submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.

- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when you define a scope in the Search References dialog box.

- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.

- **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

The following action is available from the Document > WSDL menu:

- **Show Definition** - Takes you to the location of the definition of the current item.

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

Refactoring Actions

The following refactoring actions are available from the Refactoring submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.

  *N*  Rename Component in... - Opens the Rename component type dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the Rename Component action.
Searching and Refactoring Operations Scope in WSDL Documents

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the **Change scope** operation, available in the Quick Fix action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the **Master Files support**.

WSDL Resource Hierarchy/Dependencies View in WSDL Documents

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a WSDL resource. To open this view, go to **Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies**.

**Note:** The hierarchy of a WSDL resource includes the hierarchy of imported XML Schema resources. The dependencies of an XML Schema resource present the WSDL documents that import the schema.
To view the hierarchy of a WSDL document, select the document in the project view and choose **Resource Hierarchy** from the contextual menu.

![Resource Hierarchy/Dependencies View](image)

**Figure 121: Resource Hierarchy/Dependencies View**

If you want to see the dependencies of a WSDL document, select the document in the project view and choose **Resource Dependencies** from the contextual menu.
The following actions are available in the Resource Hierarchy/Dependencies view:

- **Refresh**
  Refreshes the Hierarchy/Dependencies structure.

- **Stop**
  Stops the hierarchy/dependencies computing.

- **Show Hierarchy**
  Allows you to choose a resource to compute the hierarchy structure.

- **Show Dependencies**
  Allows you to choose a resource to compute the dependencies structure.

- **Configure**
  Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

- **History**
  Provides access to the list of previously computed dependencies. Use the **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

- **Open**
  Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

- **Copy location**
  Copies the location of the resource.

- **Move resource**
  Moves the selected resource.

- **Rename resource**
  Renames the selected resource.
Show Resource Hierarchy
Shows the hierarchy for the selected resource.

Show Resource Dependencies
Shows the dependencies for the selected resource.

Add to Master Files
Adds the currently selected resource in the Master Files directory.

Expand All
Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All
Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

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

Note: The Move resource or Rename resource actions give you the option to update the references to the resource.

Moving/Renaming WSDL Resources
You are able to move and rename a resource presented in the Resource/Hierarchy Dependencies view, using the Rename resource and Move resource refactoring actions from the contextual menu.

When you select the Rename action in the contextual menu of the Resource/Hierarchy Dependencies view, the Rename resource dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the Move action from the contextual menu of the Resource/Hierarchy Dependencies view, the Move resource dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

Component Dependencies View in WSDL Documents
The Component Dependencies view allows you to view the dependencies for a selected WSDL component. To open the Component Dependencies view, go to Window > Show View > Other > oXygen XML Editor > Component Dependencies.

To view the dependencies of an WSDL component, select the desired component in the editor and choose the Component Dependencies action from the contextual menu. This action is available for all WSDL components (messages, port types, operations, bindings and so on).

Note: If you search for dependencies of XML Schema elements, the Component Dependencies view presents the references from WSDL documents.
The following actions are available in the toolbar of the **Component Dependencies** view:

- **Refresh**
  Refreshes the dependencies structure.

- **Stop**
  Stops the dependencies computing.

- **Configure**
  Allows you to configure a *search scope* to compute the dependencies structure. You can decide to use the defined scope for future operations automatically, by checking the corresponding check box.

- **History**
  Allows you to repeat a previous dependencies computation.

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

- **Go to First Reference**
  Selects the first reference of the referenced component from the current selected component in the dependencies tree.

- **Go to Component**
  Displays the definition of the current selected component in the dependencies tree.

**Tip:** If a component contains multiple references to another, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon ♻.

### Highlight Component Occurrences in WSDL Documents

When you position your mouse cursor over a component in a WSDL document, Oxygen XML Developer plugin searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default behaviour of **Highlight Component Occurrences**, open the **Preferences** dialog box and go to **Editor > Mark Occurrences**. You can also trigger a search using the **Search > Search Occurrences in File** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.
Quick Assist Support in WSDL Documents

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

![WSDL Quick Assist Support](image)

**Figure 124: WSDL Quick Assist Support**

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

- **Rename Component in...**
  Renames the component and all its dependencies.

- **Search Declarations**
  Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

- **Search References**
  Searches all references of the component in a predefined scope.

- **Component Dependencies**
  Searches the component dependencies in a predefined scope.

- **Change Scope...**
  Configures the scope that will be used for future search or refactor operations.

- **Rename Component**
  Allows you to rename the current component in-place.

- **Search Occurrences**
  Searches all occurrences of the component within the current file.

Generating Documentation for WSDL Documents

You can use Oxygen XML Developer plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the level of details to present for each of them. Also, the components are hyperlinked.

**Note:** The WSDL documentation includes the XML Schema components that belong to the internal or imported XML schemas.

**Note:** To obtain the documentation in a custom format, *use custom stylesheets*.

To generate documentation for a WSDL document, select **WSDL Documentation...** from the **XML Tools > Generate Documentation** menu or from the **Generate WSDL Documentation** action from the contextual menu of the **Navigator** view.
The **Input URL** field of the dialog panel must contain the full path to the WSDL document that you want to generate documentation for. The WSDL document can be located either local or remote. You can also specify the path to the WSDL document using editor variables.

You can split the output into multiple files using different criteria. For large WSDL documents, choosing a different split criterion may generate smaller output files providing a faster documentation browsing.

The available split criteria are:

- **By location** - each output file contains the components from the same WSDL document.
- **By namespace** - each output file contains information about components with the same namespace.
- **By component** - each output file contains information about one WSDL or XML Schema component.
When you generate documentation for WSDL documents, you can choose what components (services, bindings, messages and others) and details (namespace, location, instance and others) to include in the documentation:

- **Components**
  - **Services** - specifies whether the generated documentation includes the WSDL services.
  - **Bindings** - specifies whether the generated documentation includes the WSDL bindings.
  - **Port Types** - specifies whether the generated documentation includes the WSDL port types.
  - **Messages** - specifies whether the generated documentation includes the WSDL messages.
  - **XML Schema Components** specifies whether the generated documentation includes the XML Schema components.
    - **Only global elements and types** - specifies whether the generated documentation includes only global elements and types.

- **Details**
  - **Namespace** - presents the namespace information for WSDL or XML Schema components.
  - **Location** - presents the location information for each WSDL or XML Schema component.
  - **Used by** - presents the list of components that reference the current one.
  - **Documentation** - presents the component documentation. In case you choose **Escape XML Content**, the XML tags are presented in the documentation.
  - **Source** - presents the XML fragment that defines the current component.
  - **Instance** - generates a sample XML instance for the current component.

  **Note**: This option applies to the XML Schema components only.

- **XML Schema Diagram** - Displays the diagram for each XML Schema component. You can choose the image format (JPEG, PNG, SVG) to use for the diagram section.
- **Diagram annotations** - specifies whether the annotations of the components presented in the diagram sections are included.
The default format of the generated WSDL documentation is HTML.

By default, the documentation of each component is presented to the right side. Each component is displayed in a separate section. The title of the section is composed of the component type and the component name. The component information (namespace, documentation and so on) is presented in a tabular form. The left side of the output holds the table of contents. The table of contents is divided in two tabs: Components and Resource Hierarchy.

The Components tab allows you to group the contents by namespace, location, or component type. The WSDL components from each group are sorted alphabetically. The Resource Hierarchy tab displays the dependencies between WSDL and XML Schema modules in a tree like fashion. The root of the tree is the WSDL document that you generate documentation for.

If you split the output in multiple files, the table of contents is displayed in the left frame. The contents are grouped using the same criteria as the split.

After the documentation is generated, you can collapse details for some WSDL components using the Showing view.

---

**Figure 127: WSDL Documentation in HTML Format**

By default, the documentation of each component is presented to the right side. Each component is displayed in a separate section. The title of the section is composed of the component type and the component name. The component information (namespace, documentation and so on) is presented in a tabular form. The left side of the output holds the table of contents. The table of contents is divided in two tabs: Components and Resource Hierarchy.

The Components tab allows you to group the contents by namespace, location, or component type. The WSDL components from each group are sorted alphabetically. The Resource Hierarchy tab displays the dependencies between WSDL and XML Schema modules in a tree like fashion. The root of the tree is the WSDL document that you generate documentation for.

If you split the output in multiple files, the table of contents is displayed in the left frame. The contents are grouped using the same criteria as the split.

After the documentation is generated, you can collapse details for some WSDL components using the Showing view.

---

**Figure 128: The Showing View**
Generating Documentation for WSDL Documents in a Custom Format

To obtain the default HTML documentation output from a WSDL document, Oxygen XML Developer plugin uses an intermediary XML document to which it applies an XSLT stylesheet. To create a custom output from your WSDL document, edit this XSLT stylesheet or write your own one.

Note: The `wsdlDocHtml.xsl` stylesheet used to obtain the HTML documentation is located in the installation folder of Oxygen XML Developer plugin, in the `[OXYGEN_DIR]/frameworks/wsdl_documentation/xsl` folder.

Note: The intermediary XML document complies with the `wsdlDocSchema.xsd` XML Schema. This schema is located in the installation folder of Oxygen XML Developer plugin, in the `[OXYGEN_DIR]/frameworks/wsdl_documentation` folder.

Figure 129: The Custom Format Options Dialog

When using a custom format, you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating Documentation for WSDL Documents from the Command Line

To generate documentation from a WSDL document from the command line, open the WSDL Documentation dialog and click Export settings. Using the exported settings file you can generate the same documentation from the command line by running the following scripts:

- `wsdlDocumentation.bat` on Windows.
- `wsdlDocumentation.sh` on Unix / Linux.
- `wsdlDocumentationMac.sh` on Mac OS X.

The scripts are located in the installation folder of Oxygen XML Developer plugin. You can integrate the scripts in an external batch process launched from the command-line interface.

WSDL SOAP Analyzer

After you edit and validate your Web service descriptor against a mix of the XML Schemas for WSDL and SOAP, it is easy to check if the defined SOAP messages are accepted by the remote Web Services server using Oxygen XML Developer plugin's WSDL SOAP Analyser integrated tool.

Composing a SOAP Request

WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

Oxygen XML Developer plugin provides two ways of testing, one for the currently edited WSDL document and another for the remote WSDL documents that are published on a web server. To open the WSDL SOAP Analyser tool for the currently edited WSDL document do one of the following:

- Click the WSDL SOAP Analyser toolbar button.
- Go to WSDL > WSDL SOAP Analyser.
• Go to in the contextual menu of the **Navigator** view.

![Figure 130: WSDL SOAP Analyser](image)

This dialog box contains a SOAP analyser and sender for Web Services Description Language file types. The analyser fields are:

- **Services** - The list of services defined by the WSDL file.
- **Ports** - The ports for the selected service.
- **Operations** - The list of available operations for the selected service.
- **Action URL** - Shows the script that serves the operation.
- **SOAP Action** - Identifies the action performed by the script.
- **Version** - Choose between 1.1 and 1.2. The SOAP version is selected automatically depending on the selected port.
- **Request Editor** - It allows you to compose the web service request. When an action is selected, Oxygen XML Developer plugin tries to generate as much content as possible for the SOAP request. The envelope of the SOAP request has the correct namespace for the selected SOAP version, that is `http://schemas.xmlsoap.org/soap/envelope/` for SOAP 1.1 or `http://www.w3.org/2003/05/soap-envelope` for SOAP 1.2. Usually you just have to change few values in order for the request to be valid. The content completion assistant is available for this editor and is driven by the schema that defines the type of the current message. While selecting different operations, Oxygen XML Developer plugin remembers the modified request for each one. You can press the **Regenerate** button in order to overwrite your modifications for the current request with the initial generated content.
- **Attachments List** - You can define a list of file URLs to be attached to the request.
- **Response Area** - Initially it displays an auto generated server sample response so you can have an idea about how the response looks like. After pressing the **Send** button, it presents the message received from the server in response to the Web Service request. It may show also error messages. In case the response message contains attachments, Oxygen XML Developer plugin prompts you to save them, then tries to open them with the associated system application.
- **Errors List** - There may be situations in which the WSDL file is respecting the WSDL XML Schema, but it fails to be valid for example in the case of a message that is defined by means of an element that is not found in the types section of the WSDL. In such a case, the errors are listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog box is displayed when Oxygen XML Developer plugin is connecting to the server.

The testing of a WSDL file is straight-forward: click the WSDL analysis button, then select the service, the port, and the operation. The editor generates the skeleton for the SOAP request. You can edit the request, eventually attach files to it and send it to the server. Watch the server response in the response area. You can find more details in the **Testing Remote WSDL Files** section.

**Note:** SOAP requests and responses are automatically validated in the **WSDL SOAP Analyser** using the XML Schemas specified in the WSDL file.
Once defined, a request derived from a Web Service descriptor can be saved with the Save button to a Web Service SOAP Call (WSSC) file for later reuse. In this way, you save time in configuring the URLs and parameters.

You can open the result of a Web Service call in an editor panel using the Open button.

Testing Remote WSDL Files

To open and test a remote WSDL file the steps are the following:

1. Go to menu Window > Show View > Other > oXygen > WSDL SOAP Analyser ....
2. Press the Choose WSDL button and enter the URL of the remote WSDL file.
   You enter the URL:
   • by typing
   • by browsing the local file system
   • by browsing a remote file system
   • by browsing a UDDI Registry
3. Press the OK button.
   This will open the WSDL SOAP Analyser tool. In the Saved SOAP Request tab you can open directly a previously saved Web Service SOAP Call (WSSC) file thus skipping the analysis phase.

The UDDI Registry Browser

Pressing the button in the WSDL File Opener dialog (menu Tools > WSDL SOAP Analyzer) opens the UDDI Registry Browser dialog.

![UDDI Registry Browser dialog](image)

Figure 131: UDDI Registry Browser dialog
The fields of the dialog are the following:

- **URL** - Type the URL of an UDDI registry or choose one from the default list.
- **Keywords** - Enter the string you want to be used when searching the selected UDDI registry for available Web services.
- **Rows to fetch** - The maximum number of rows to be displayed in the result list.
- **Search by** - You can choose to search either by company or by provided service.
- **Case sensitive** - When checked, the search takes into account the keyword case.
- **Search** - The WSDL files that matched the search criteria are added in the result list.

When you select a WSDL from the list and click the **OK** button, the **UDDI Registry Browser** dialog is closed and you are returned to the WSDL File Opener dialog.

---

**Editing CSS Stylesheets**

This section explains the features of the editor for CSS stylesheets and how these features should be used.

**Validating CSS Stylesheets**

Oxygen XML Developer plugin includes a built-in **CSS Validator**, integrated with general validation support. This makes the usual validation features for presenting errors also available for CSS stylesheets.

The CSS properties accepted by the validator are those included in the current CSS profile that is selected in *the CSS validation preferences*. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties and the CSS extensions specific for Oxygen. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.

**Specify Custom CSS Properties**

Lists the steps required for specifying custom CSS properties.

To specify custom CSS properties, follow these steps:

1. Create a file named **CustomProperties.xml** that has the following structure:

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <css_keywords
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://www.oxygenxml.com/ns/css">
     <property name="custom">
       <summary>Description for custom property.</summary>
       <value name="customValue"/>
       <value name="anotherCustomValue"/>
     </property>
   </css_keywords>
   ```

2. Go to your desktop and create the `builtin/css-validator/` folder structure.
3. Press and hold **Shift** and right-click on your desktop. From the contextual menu, select **Open Command Window Here**.
4. In the command line, run the `jar cvf custom_props.jar builtin/command`. The `custom_props.jar` file is created.
5. Go to `[OXYGEN_DIR]/lib` and create the `endorsed` folder. Copy the `custom_props.jar` file to `[OXYGEN_DIR]/lib/endorsed`.

**Content Completion in CSS Stylesheets**

A **Content Completion Assistant**, similar to *the one available for XML documents* offers the CSS properties and the values available for each property. It is activated with the **Ctrl Space (Command Space on OS X)** shortcut and is context-sensitive when invoked for the value of a property. The **Content Completion Assistant** also includes **code templates that can be used to quickly insert code fragments** into CSS stylesheets. The code templates that are proposed include form controls, actions, and **Author** mode operations.
The properties and values available are dependent on the CSS Profile selected in the CSS preferences. The CSS 2.1 set of properties and property values is used for most of the profiles. However, with CSS 1 and CSS 3 specific proposal sets are used.

**CSS Outline View**

The CSS Outline view presents the import declarations for other CSS stylesheet files and all the selectors defined in the current CSS document. The selector entries can be presented as follows:

- in the order they appear in the document
- sorted by the element name used in the selector
- sorted by the entire selector string representation

You can synchronize the selection in the Outline view with the caret moves or changes you make in the stylesheet document. When you select an entry from the Outline view, Oxygen XML Developer plugin highlights the corresponding import or selector in the CSS editor.

The selectors presented in this view can be found quickly using the key search field. When you press a sequence of character keys while the focus is in the view, the first selector that starts with that sequence is selected automatically.

**Folding in CSS Stylesheets**

In a large CSS stylesheet document, some styles can be collapsed so that only the styles that are needed remain in focus. The same folding features available for XML documents are also available in CSS stylesheets.

*Note:* To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.
Formatting and Indenting CSS Stylesheets (Pretty Print)

If the edited CSS stylesheet becomes unreadable because of the bad alignment of the text lines, the format and indent operation available for XML documents is also available for CSS stylesheets. It works in the same way as for XML documents and is available as the same menu and toolbar action.

Minifying CSS Stylesheets

Minification (or compression) of a CSS document is the practice of removing unnecessary code without affecting the functionality of the stylesheet.

To minify a CSS, invoke the contextual menu anywhere in the edited document and choose the Minify CSS... action. Oxygen XML Developer plugin opens a dialog box that allows you to:

- Set the location of the resulting CSS.
- Place each style rule on a new line.

After pressing OK, Oxygen XML Developer plugin performs the following actions:

- All spaces are normalized (all leading and trailing spaces are removed, while sequences of white spaces are replaced with single space characters).
- All comments are removed.

Note: The CSS minifier relies heavily upon the W3C CSS specification. If the content of the CSS file you are trying to minify does not conform with the specifications, an error dialog box will be displayed, listing all errors encountered during the processing.

The resulting CSS stylesheet gains a lot in terms of execution performance, but loses in terms of readability. The source CSS document is left unaffected.

Note: To restore the readability of a minified CSS, invoke the Format and Indent action from the XML menu, the Source submenu from the contextual menu, or Source toolbar. However, this action will not recover any of the deleted comments.

Other CSS Editing Actions

The CSS editor type offers a reduced version of the popup menu available in the XML editor. Only the folding actions, the edit actions and a part of the source actions (only the actions To lower case, To upper case, Capitalize lines) are available.

Editing LESS CSS Stylesheets

Oxygen XML Developer plugin provides support for stylesheets coded with the LESS dynamic stylesheet language. LESS extends the CSS language by adding features that allow mechanisms such as variables, nesting, mixins, operators, and functions. Oxygen XML Developer plugin offers additional LESS features that include:

- Open LESS files - the LESS extension is recognized and thus can be opened by the editor
- Validation - presents errors in LESS files
- Content completion - offers properties and the values available for each property
- Compile to CSS - options are available to compile LESS files to CSS

Note: Oxygen XML Developer plugin also support syntax highlighting in LESS files, although there may be some limitations in supporting all the LESS constructs.

For more information about LESS go to http://lesscss.org/.
Validating LESS Stylesheets

Oxygen XML Developer plugin includes a built-in **LESS CSS Validator**, integrated with general validation support. The usual validation features for presenting errors also available for LESS stylesheets.

Oxygen XML Developer plugin provides three validation methods:

- Automatic validation as you type - marks validation errors in the document as you are editing.
- Validation upon request, by pressing the Validate button from the Validation toolbar drop-down list. An error list is presented in the message panel at the bottom of the editor.
- Validation scenarios, by selecting Configure Validation Scenario(s)... from the Validation toolbar drop-down list. Errors are presented in the message panel at the bottom of the editor. This is useful when you need to validate the current file as part of a larger LESS import hierarchy (for instance, you may change the URL of the file to validate to the root of the hierarchy).

Content Completion in LESS Stylesheets

A **Content Completion Assistant** offers the LESS properties and the values available for each property. It is activated with the Ctrl Space (Command Space on OS X) shortcut and is context-sensitive when invoked for the value of a property in a LESS file. The **Content Completion Assistant** also includes code templates that can be used to quickly insert code fragments into LESS stylesheets. The code templates that are proposed include form controls, actions, and Author mode operations.

![Content Completion in LESS Stylesheets](image)

**Figure 134: Content Completion in LESS Stylesheets**

The properties and values available are dependent on the CSS Profile selected in the CSS preferences.

Compiling LESS Stylesheets to CSS

When editing LESS files, you can compile the files into CSS. Oxygen XML Developer plugin provides both manual and automatic options to compile LESS stylesheets into CSS.

You have two options for compiling LESS files to CSS:

1. Use the contextual menu in a LESS file and select Compile to CSS (Ctrl Shift C (Command Shift C on OS X)).
2. Enable the option Automatically compile LESS to CSS when saving in the settings. To do so, open the Preferences dialog box and go to Editor > Open > Save > Save hooks. If enabled, when you save a LESS file it will automatically be compiled to CSS (this option is disabled by default).

   **Important:** If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

Editing Relax NG Schemas

Oxygen XML Developer plugin provides a special type of editor for Relax NG schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the standard outline mode and the components mode.
Editing Relax NG Schema in the Master Files Context

Smaller interrelated modules that define a complex Relax NG Schema cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, an element defined in a main schema document is not visible when you edit an included module. Oxygen XML Developer plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You can set a main Relax NG document either using the master files support from the Navigator view, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Developer plugin warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

• correct validation of a module in the context of a larger schema structure;
• Content Completion Assistant displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one;
• the Outline displays the components collected from the entire schema structure;

Relax NG Schema Diagram

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

Introduction

Oxygen XML Developer plugin provides a simple, expressive, and easy to read Schema Diagram view for Relax NG schemas.

With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, or BMP images. It helps both schema authors in developing the schema and content authors who are using the schema to understand it.

Oxygen XML Developer plugin is the only XML editor to provide a side by side source and diagram presentation and have them real-time synchronized:

• the changes you make in the Editor are immediately visible in the Diagram (no background parsing);
• changing the selected element in the diagram selects the underlying code in the source editor.

Full Model View

When you create a new schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The Diagram view has two tabbed panes offering a Full Model View and a Logical Model View.
The following references can be expanded in place: patterns, includes, and external references. This expansion mechanism, coupled with the synchronization support, makes the schema navigation easy.

All the element and attribute names are editable: double-click any name to start editing it.

Symbols Used in the Schema Diagram

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

- - define pattern with the name attribute set to the value shown inside the rectangle (in this example name).
- - define pattern with the combine attribute set to *interleave* and the name attribute set to the value shown inside the rectangle (in this example attlist.person).
- - define pattern with the combine attribute set to *choice* and the name attribute set to the value shown inside the rectangle (in this example attlist.person).
- - element pattern with the name attribute set to the value shown inside the rectangle (in this example name).
- - attribute pattern with the name attribute set to the value shown inside the rectangle (in this case note).
- - ref pattern with the name attribute set to the value shown inside the rectangle (in this case family).
- oneOrMore pattern.
- zeroOrMore pattern.
- optional pattern.
- choice pattern.
- value pattern, used for example inside a choice pattern.
- group pattern.
- pattern from the Relax NG Annotations namespace (http://relaxng.org/ns/compatibility/annotations/1.0) which is treated as a documentation element in a Relax NG schema.
- text pattern.
- empty pattern.

**Logical Model View**

The **Logical Model View** presents the compiled schema which is a single pattern. The patterns that form the element content are defined as top level patterns with generated names. These names are generated depending of the elements name class.

![Logical Model View for a Relax NG Schema](image)
**Actions Available in the Diagram View**

The contextual menu offers the following actions:

**Append child**
- Appends a child to the selected component.

**Insert Before**
- Inserts a component before the selected component.

**Insert After**
- Inserts a component after the selected component.

**Edit attributes**
- Edits the attributes of the selected component.

**Remove**
- Removes the selected component.

**Show only the selected component**
- Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.

**Show Annotations**
- Depending on its state (selected/not selected), the documentation nodes are shown or hidden.

**Auto expand to references**
- This option controls how the schema diagram is automatically expanded. If you select it and then edit a top-level element or you make a refresh, the diagram is expanded until it reaches referenced components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.

**Collapse Children**
- Collapses the children of the selected view.

**Expand Children**
- Expands the children of the selected view.

**Print Selection...**
- Prints the selected view.

**Save as Image...**
- Saves the current selection as JPEG, BMP, SVG or PNG image.

**Refresh**
- Refreshes the schema diagram according to the changes in your code. They represent changes in your imported documents or changes that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

**Relax NG Outline View**

The Relax NG Outline view presents a list with the patterns that appear in the diagram in both the **Full Model View** and **Logical Model View** cases. It allows a quick access to a component by name. By default it is displayed on screen. If you closed the Outline view you can reopen it from menu **Window > Show View > Other > oXygen > Outline**. You can switch between the Relax NG patterns version and the **standard XML version** of the view by pressing the ![Show components/Show XML structure](image.png) button.
Figure 137: Relax NG Outline View

The tree shows the XML structure or the define patterns collected from the current document. By default, the Outline view presents the define patterns.

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

- **Selection update on caret move**
  Allows a synchronization between Outline view and schema diagram. The selected view from the diagram will be also selected in the Outline view.

- **Show components**
  Shows the define patterns collected from the current document.

- **Flat presentation mode of the filtered results**
  When active, the application flattens the filtered result elements to a single level.

- **Show comments and processing instructions**
  Show/hide comments and processing instructions in the Outline view.

- **Show element name**
  Show/hide element name.

- **Show text**
  Show/hide additional text content for the displayed elements.

- **Show attributes**
  Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from the Outline preferences panel.

- **Configure displayed attributes**
  Displays the XML Structured Outline preferences page.

When components are displayed, the following action is available in the View menu on the Outline view's action bar:
Show XML structure

Shows the XML structure of the current document.

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

Relax NG Editor Specific Actions

The list of actions specific for the Relax NG (full syntax) editor is:

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

Searching and Refactoring Actions in RNG Schemas

Search Actions

The following search actions can be applied on named defines and are available from the Search submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.

- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the Search References dialog box.

- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.

- **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on named defines and are available from the Refactoring submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the Esc or Enter key on your keyboard.

- **Rename Component in...** - Opens the Rename component_type dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the Preview button, you can view the files affected by the Rename Component action.
**RNG Resource Hierarchy/Dependencies View**

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a schema. To open this view, go to **Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies**.

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

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

- **Refresh**
  Refreshes the Hierarchy/Dependencies structure.

- **Stop**
  Stops the hierarchy/dependencies computing.

- **Show Hierarchy**
  Allows you to choose a resource to compute the hierarchy structure.

- **Show Dependencies**
  Allows you to choose a resource to compute the dependencies structure.

- **Configure**
  Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

- **History**
  Provides access to the list of previously computed dependencies. Use the ✗ Clear history button to remove all items from this list.

The contextual menu contains the following actions:

- **Open**
  Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

- **Copy location**
  Copies the location of the resource.

- **Move resource**
  Moves the selected resource.

- **Rename resource**
  Renames the selected resource.
**Show Resource Hierarchy**
Shows the hierarchy for the selected resource.

**Show Resource Dependencies**
Shows the dependencies for the selected resource.

**Add to Master Files**
Adds the currently selected resource in the Master Files directory.

**Expand All**
Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

**Collapse All**
Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

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

**Note:** The **Move resource** or **Rename resource** actions give you the option to update the references to the resource.

### Moving/Renaming RNG Resources
You are able to move and rename a resource presented in the Resource/Hierarchy Dependencies view, using the Rename resource and Move resource refactoring actions from the contextual menu.

When you select the Rename action in the contextual menu of the Resource/Hierarchy Dependencies view, the Rename resource dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the Move action from the contextual menu of the Resource/Hierarchy Dependencies view, the Move resource dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

**Note:** Updating the references of a resource that is resolved through a catalog is not supported. Also, the update references operation is not supported in case the path to the renamed or moved resource contains entities.

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

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

- **Refresh**
  Refreshes the dependencies structure.

- **Stop**
  Stops the dependencies computing.

- **Configure**
  Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

- **History**
  Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

- **Go to First Reference**
  Selects the first reference of the referenced component from the current selected component in the dependencies tree.

- **Go to Component**
  Shows the definition of the current selected component in the dependencies tree.

**Tip:** If a component contains multiple references to another components, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon ⊗.
**RNG Quick Assist Support**

The Quick Fix support improves the development work flow, offering fast access to the most commonly used actions when you edit XML Schema documents.

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

![Image of Quick Assist Support](image)

**Figure 142: RNG Quick Assist Support**

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

- **Rename Component in...**
  Renames the component and all its dependencies.

- **Search Declarations**
  Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

- **Search References**
  Searches all references of the component in a predefined scope.

- **Component Dependencies**
  Searches the component dependencies in a predefined scope.

- **Change Scope...**
  Configures the scope that will be used for future search or refactor operations.

- **Rename Component**
  Allows you to rename the current component in-place.

- **Search Occurrences**
  Searches all occurrences of the component within the current file.

**Configuring a Custom Datatype Library for a RELAX NG Schema**

A RELAX NG schema can declare a custom datatype library for the values of elements found in XML document instances. The datatype library must be developed in Java and it must implement the interface specified on the [www.thaiopensource.com website](http://www.thaiopensource.com).

The jar file containing the custom library and any other dependent jar file must be added to the classpath of the application, that is the jar files must be added to the folder `[OXYGEN_PLUGIN_DIR]/lib` and a line `<library name="lib/custom-library.jar"/>` must be added for each jar file to the file `[OXYGEN_PLUGIN_DIR]/plugin.xml` in the `<runtime>` element.

To load the custom library, restart the Eclipse platform.
Editing NVDL Schemas

Some complex XML documents are composed by combining elements and attributes from different namespaces. More, the schemas that define these namespaces are not even developed in the same schema language. In such cases, it is difficult to specify in the document all the schemas which must be taken into account for validation of the XML document or for content completion. An NVDL (Namespace Validation Definition Language) schema can be used. This schema allows the application to combine and interleave multiple schemas of different types (W3C XML Schema, RELAX NG schema, Schematron schema) in the same XML document.

Oxygen XML Developer plugin provides a special type of editor for NVDL schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the standard outline mode and the components mode.

NVDL Schema Diagram

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

Introduction

Oxygen XML Developer plugin provides a simple, expressive, and easy to read Schema Diagram View for NVDL schemas.

With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, and BMP images. It helps both schema authors in developing the schema and content authors that are using the schema to understand it.

Oxygen XML Developer plugin is the only XML Editor to provide a side by side source and diagram presentation and have them real-time synchronized:

• the changes you make in the Editor are immediately visible in the Diagram (no background parsing).
• changing the selected element in the diagram, selects the underlying code in the source editor.

Full Model View

When you create a schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The diagram view has two tabbed panes offering a Full Model View and a Logical Model View.
The **Full Model View** renders all the NVDL elements with intuitive icons. This representation coupled with the synchronization support makes the schema navigation easy.

Double click on any diagram component in order to edit its properties.

### Actions Available in the Diagram View

The contextual menu offers the following actions:

- **Show only the selected component**
  - Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.

- **Show Annotations**
  - Depending on its state (selected/not selected), the documentation nodes are shown or hidden.

- **Auto expand to references**
  - This option controls how the schema diagram is automatically expanded. For instance, if you select it and then edit a top-level element or you trigger a diagram refresh, the diagram will be expanded until it reaches the referenced components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.

- **Collapse Children**
  - Collapses the children of the selected view.
Expand Children

Expands the children of the selected view.

Print Selection...

Prints the selected view.

Save as Image...

Saves the current selection as image, in JPEG, BMP, SVG or PNG format.

Refresh

Refreshes the schema diagram according to the changes in your code (changes in your imported documents or those that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the Logical Model View instead of the diagram.

NVDL Outline View

The NVDL Outline view presents a list with the named or anonymous rules that appear in the diagram. It allows a quick access to a rule by name. It can be opened from the Window > Show View > Other > oXygen XML Editor > Outline menu.

NVDL Editor Specific Actions

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

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

Searching and Refactoring Actions in NVDL Schemas

Search Actions

The following search actions can be applied on name, useMode, and startMode attributes and are available from the Search submenu in the contextual menu of the current editor:

- ![Search References](image) - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.

- ![Search References in...](image) - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the Search References dialog box.

- ![Search Declarations](image) - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

- ![Search Declarations in...](image) - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.

- ![Search Occurrences in File](image) - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on name, useMode, and startMode attributes and are available from the Refactoring submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the Esc or Enter key on your keyboard.
• **Rename Component in...** - Opens the **Rename component_type** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

![figure](image)

**Figure 144: Rename Identity Constraint Dialog Box**

### Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected NVDL named mode. You can open the view from **Window > Show View > Other > <oXygen/> XML > Component Dependencies**.

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

![figure](image)

**Figure 145: Component Dependencies View - Hierarchy for test.nvdl**

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

- **Refresh**
  
  Refreshes the dependencies structure.
Stop
Allows you to stop the dependencies computing.

Configure
Allows you to configure a search scope to compute the dependencies structure. If you decide to set the application to use automatically the defined scope for future operations, select the corresponding checkbox.

History
Repeats a previous dependencies computation.

The following actions are available in the contextual menu:

Go to First Reference
Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component
Shows the definition of the current selected component in the dependencies tree.

Tip: If a component contains multiple references to another component, a small table containing all references is shown. When a recursive reference is encountered it is marked with a special icon

---

Editing JSON Documents

This section explains the features of the Oxygen XML Developer plugin JSON Editor and how to use them.

JSON Editor Text Mode

The Text Mode of the JSON editor provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, drag and drop, and other editor actions like validation and formatting and indenting (pretty print) document.

You can use the two Text and Grid buttons available at the bottom of the editor panel if you want to switch between the editor Text Mode and Grid Mode.
Syntax highlight in JSON Documents

Oxygen XML Developer plugin supports Syntax Highlight for JavaScript / JSON editors and provides default configurations for the JSON set of tokens. You can customize the foreground color, background color and the font style for each JSON token type.

Folding in JSON

In a large JSON document, the data enclosed in the '{' and '}' characters can be collapsed so that only the needed data remain in focus. The folding features available for XML documents are available in JSON documents.
Oxygen XML Developer plugin allows you to view and edit the JSON documents in the Grid Mode. The JSON is represented in Grid mode as a compound layout of nested tables in which the JSON data and structure can be easily manipulated with table-specific operations or drag and drop operations on the grid components. You can also use the following JSON-specific contextual actions:

**Array**
- Useful when you want to convert a JSON value to array.

**Insert value before**
- Inserts a value before the currently selected one.

**Insert value after**
- Inserts a value after the currently selected one.

**Append value as child**
- Appends a value as a child of the currently selected value.

You can customize the JSON grid appearance according to your needs. For instance you can change the font, the cell background, foreground, or even the colors from the table header gradients. The default width of the columns can also be changed.
JSON Outline View

The JSON Outline view displays the list of all the components of the JSON document you are editing. To enable the JSON Outline view, go to Window > Show view > Outline.

![JSON Outline View](image)

Figure 148: The JSON Outline View

The following actions are available in the contextual menu of the JSON Outline view:

- Cut
- Copy
- Paste
- Delete

The settings menu of the JSON Outline view allows you to enable Selection update on caret move. This option controls the synchronization between the Outline view and source the document. Oxygen XML Developer plugin synchronizes the selection in the Outline view with the caret moves or the changes you make in the JSON editor. Selecting one of the components from the Outline view also selects the corresponding item in the source document.

Validating JSON Documents

Oxygen XML Developer plugin includes a built-in JSON validator (based on the free JAVA source code available on www.json.org), integrated with the general validation support.

Convert XML to JSON

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

1. Go to menu XML Tools > XML to JSON....

The XML to JSON dialog box is displayed:
2. Choose or enter the **Input URL** of the XML document.

3. Choose the **Output file** that will contain the conversion JSON result.

4. Check the **Open in Editor** option to open the JSON result of the conversion in the Oxygen XML Developer plugin JSON Editor.

5. Click the **OK** button.

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

---

### Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Developer plugin supports StratML Part 1 (Strategic Plan) and StratML Part 2 (Performance Plans and Reports) and provides templates for the following documents:

- **Strategic Plan** (StratML Part 1)
- **Performance Plan** (StratML Part 2)
- **Performance Report** - (StratML Part 2)
• **Strategic Plan** - (StratML Part 2)

You can view the components of a StratML document in the **Outline** view. Oxygen XML Developer plugin implements a default XML with XSLT transformation scenario for this document type, called StratML to HTML.

---

## Editing JavaScript Documents

This section explains the features of the Oxygen XML Developer plugin JavaScript Editor and how you can use them.

### JavaScript Editor Text Mode

Oxygen XML Developer plugin allows you to create and edit JavaScript files and assists you with useful features such as syntax highlight, content completion, and outline view. To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.

```javascript
function newPage(filename, overlay) {
    divs = document.getElementsByTagName("div");

    if (divs) {
        var xdiv = divs[0];

        if (xdiv) {
            var xid = xdiv.getAttribute("id");

            var mytoc = window.top.frames[0];
            if (mytoc.lastUnderlined) {
                mytoc.lastUnderlined.style.textDecoration = "none";
            }

            var tdiv = xhGetElementByJId(xid, mytoc);

            if (tdiv) {
                var ta = tdiv.getElementsByTagName("a").item(0);
                ta.style.textDecoration = "underline";
                mytoc.lastUnderlined = ta;
            }
        }
    }

    if (overlay != 0) {
        overlaySetup('loc');
    }
}
```

---

**Figure 149: JavaScript Editor Text Mode**

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

- **Cut**
  - Allows you to cut fragments of text from the editing area.

- **Copy**
  - Allows you to copy fragments of text from the editing area.

- **Paste**
  - Allows you to paste fragments of text in the editing area.

- **Toggle comment**
  - Allows you to comment a line or a fragment of the JavaScript document you are editing. This option inserts a single comment for the entire fragment you want to comment.
### Toggle line comment

Allows you to comment a line or a fragment of the JavaScript document you are editing. This option inserts a comment for each line of the fragment you want to comment.

### Go to matching bracket

Use this option to find the closing, or opening bracket, matching the bracket at the caret position. When you select this option, Oxygen XML Developer plugin moves the caret to the matching bracket, highlights its row, and decorates the initial bracket with a rectangle.

**Note:** A rectangle decorates the opening, or closing bracket which matches the current one at all times.

### Compare

Select this option to open the **Diff Files** dialog and compare the file you are editing with a file you choose in the dialog.

### Open

Allows you to select one of the following actions:

- **Open File at Caret** - select this action to open the source of the file located at the caret position
- **Open File at Caret in System Application** - select this action to open the source of the file located at the caret position with the application that the system associates with the file

### Folding

Allows you to select one of the following actions:

- **Toggle Fold**
  
  Toggles the state of the current fold.

- **Collapse Other Folds (Ctrl (Meta on Mac OS)+NumPad /)**
  
  Folds all the elements except the current element.

- **Collapse Child Folds (Ctrl (Meta on Mac OS)+NumPad .)**
  
  Folds the elements indented with one level inside the current element.

- **Expand Child Folds**
  
  Unfolds all child elements of the currently selected element.

- **Expand All (Ctrl (Meta on Mac OS)+NumPad *)**
  
  Unfolds all elements in the current document.

### Source

Allows you to select one of the following actions:

- **To Lower Case**
  
  Converts the selection content to lower case characters.

- **To Upper Case**
  
  Converts the selection content to upper case characters.

- **Capitalize Lines**
  
  Converts to upper case the first character of every selected line.

- **Join and Normalize Lines**
  
  Joins all the rows you select to one row and normalizes the content.

- **Insert new line after**
  
  Inserts a new line after the line at the caret position.
Content Completion in JavaScript Files

When you edit a JavaScript document, the **Content Completion Assistant** presents you a list of the elements you can insert at the caret position. For an enhanced assistance, JQuery methods are also presented. The following icons decorate the elements in the content completion list of proposals depending on their type:

- \( f \) - function
- \( v \) - variable
- \( o \) - object
- \( p \) - property
- \( m \) - method

**Note:** These icons decorate both the elements from the content completion list of proposals and from the **Outline** view.

```javascript
function newPage(filename, overlay) {
    var divs = document.getElementsByTagName("div");
    if (divs) {
        var xdiv = divs[0];
        if (xdiv) {
            var xid = xdiv.getElementById("xid");
            if (xid) {
                var mytoc = xid.getElementsByTagName("ul");
                if (mytoc) {
                    mytoc.className = "mytoc";
                    mytoc.onmouseover = function () {
                        this.className = "mytoc-hover";
                    };
                    mytoc.onmouseout = function () {
                        this.className = "";
                    };
                    var tdiv = xdiv.getElementsByTagName("a");
                    if (tdiv) {
                        var ta = tdiv[0].getElementsByTagName("a")[0];
                        ta.style.textDecoration = "underline";
                        mytoc.onmouseover = function () {
                            this.style.textDecoration = "underline";
                        };
                        mytoc.onmouseout = function () {
                            this.style.textDecoration = "";
                        };
                        overlaySetup("lo";)
                    }
                }
            }
        }
    }
}
```

**Figure 150: JavaScript Content Completion Assistant**

The **Content Completion Assistant** collects:

- Method names from the current file and from the library files.
- Functions and variables defined in the current file.

In case you edit the content of a function, the content completion list of proposals contains all the local variables defined in the current function, or in the functions that contain the current one.

**JavaScript Outline View**

Oxygen XML Developer plugin present a list of all the components of the JavaScript document you are editing in the **Outline** view. To open the **Outline** view, go to **Window > Show View > Outline**.
The following icons decorate the elements in the **Outline** view depending on their type:

- \( f \) - function
- \( v \) - variable
- \( o \) - object
- \( p \) - property
- \( m \) - method

The contextual menu of the JavaScript **Outline** view contains the usual **Cut**, **Copy**, **Paste**, and **Delete** actions. From the settings menu, you can enable the Update selection on caret move option to synchronize the **Outline** view with the editing area.

**Validating JavaScript Files**

You have the possibility to validate the JavaScript document you are editing. Oxygen XML Developer plugin uses the Mozilla Rhino library for validation. For more information about this library, go to [http://www.mozilla.org/rhino/doc.html](http://www.mozilla.org/rhino/doc.html). The JavaScript validation process checks for errors in the syntax. Calling a function that is not defined is not treated as an error by the validation process. The interpreter discovers this error when executing the faulted line. Oxygen XML Developer plugin can validate a JavaScript document both on-request and automatically.

**Editing XProc Scripts**

An XProc script is edited as an XML document that is validated against a RELAX NG schema. If the script has an associated transformation scenario, then the XProc engine from the scenario is invoked as validating engine. The default engine for XProc scenarios is the Calabash engine which comes bundled with Oxygen XML Developer plugin version 17.0.

The content completion inside the element **input/inline** from the XProc namespace [http://www.w3.org/ns/xproc](http://www.w3.org/ns/xproc) offers elements from the following schemas depending both on the **port** attribute and the parent of the **input** element. When invoking the content completion inside the XProc element **inline**, the **Content Completion Assistant**’s proposals list is populated as follows:

- If the value of the **port** attribute is **stylesheet** and the **xslt** element is the parent of the **input** elements, the **Content Completion Assistant** offers XSLT elements.
• If the value of the `port` attribute is `schema` and the `validate-with-relax-ng` element is the parent of the `input` element, the **Content Completion Assistant** offers RELAX NG schema elements.

• If the value of the `port` attribute is `schema` and the `validate-with-xml-schema` element is the parent of the `input` element, the **Content Completion Assistant** offers XML Schema schema elements.

• If the value of the `port` attribute is `schema` and the `validate-with-schematron` element is the parent of the `input` element, the **Content Completion Assistant** offers either ISO Schematron elements or Schematron 1.5 schema elements.

• If the above cases do not apply, then the **Content Completion Assistant** offers elements from all the schemas from the above cases.

The XProc editor assists you in writing XPath expressions by offering a **Content Completion Assistant** and dedicated coloring schemes. To customize the coloring schemes, open the **Preferences** dialog box and go to **Syntax Highlight**.

![Figure 152: XProc Content Completion](image)

### Editing Schematron Schemas

Schematron is a simple and powerful Structural Schema Language for making assertions about patterns found in XML documents. It relies almost entirely on XPath query patterns for defining rules and checks. Schematron validation rules allow you to specify a meaningful error message. This error message is provided to you if an error is encountered during the validation stage.

The Skeleton XSLT processor is used for validation and conforms with ISO Schematron or Schematron 1.5. It allows you to validate XML documents against Schematron schemas or against combined RELAX NG / W3C XML Schema and Schematron.

Oxygen XML Developer plugin assists you in editing Schematron documents with schema-based content completion, syntax highlight, search and refactor actions, and dedicated icons for the **Outline** view. You can create a new Schematron schema using one of the Schematron templates available in the New Document wizard.

The Schematron editor renders with dedicated coloring schemes the XPath expressions. To customize the coloring schemes, open the **Preferences** dialog box and go to Editor > Syntax Highlight.

**Note:** When you create a Schematron document, Oxygen XML Developer plugin provides a built-in transformation scenario. You are able to use this scenario to obtain the XSLT style-sheet corresponding to the Schematron schema. You can apply this XSLT stylesheet to XML documents to obtain the Schematron validation results.
Validate an XML Document

To validate an XML document against a Schematron schema, select the Validate action from the Validation toolbar drop-down list, the XML menu, or from the Validate menu when invoking the contextual menu in the Navigator view. If you would like to add a persistence association between your Schematron rules and the current edited XML document, use the Associate Schema action from the Document > Schema menu or the Document toolbar. A custom processing instruction is added into the document and the validation process will take into account the Schematron rules:

```xml
<?xml-model href="percent.sch" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"/>
```

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

To set a severity level, Oxygen XML Developer plugin looks for the following information:

- The role attribute, which can have one of the following values:
  - warn or warning, to set the severity level to warning
  - error, to set the severity level to error
  - fatal, to set the severity level to fatal
  - info or information, to set the severity level to info

- The start of the message, after trimming leading white-spaces. Oxygen XML Developer plugin looks to match the following exact string of characters (case sensitive):
  - Warning:, to set the severity level to warning
  - Error:, to set the severity level to error
  - Fatal:, to set the severity level to fatal
  - Info:, to set the severity level to info

  Note: Displayed message does not contain the matched prefix.

- If none of the previous rules match, Oxygen XML Developer plugin sets the severity level to error.

Validating Schematron Documents

By default, a Schematron schema is validated as you type. To change this, open the Preferences dialog box, go to Editor > Document Checking, and disable the Enable automatic validation option.

To validate your Schematron document manually, select the Validate action from the Validation toolbar drop-down list or the XML menu. When Oxygen XML Developer plugin validates a Schematron schema, it expands all the included modules so the entire schema hierarchy is validated. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

Oxygen XML Developer plugin offers an error management mechanism capable of pinpointing errors in XPath expressions and in the included schema modules.

Content Completion in Schematron Documents

Oxygen XML Developer plugin helps you edit a Schematron schema, offering, through the Content Completion Assistant, items that are valid at the caret position. When you edit the value of an attribute that refers a component, the proposed components are collected from the entire schema hierarchy. For example, if the editing context is phase/active/@pattern, the Content Completion Assistant proposes all the defined patterns.

Note: For Schematron resources, the Content Completion Assistant collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the Master Files support go to Defining Master Files at Project Level.
If the editing context is an attribute value that is an XPath expression (like `assert/@test` or `report/@test`), the Content Completion Assistant offers the names of XPath functions, the XPath axes, and user-defined variables.

The **Content Completion Assistant** displays XSLT 1.0 functions and optionally XSLT 2.0/3.0 functions in the attributes `path`, `select`, `context`, `subject`, `test` depending on the Schematron options that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.6.0.5 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion also displays the XSLT Saxon extension functions as in the following figure:

![XSLT extension functions in Schematron schemas documents](image)

**Figure 153: XSLT extension functions in Schematron schemas documents**

The **Content Completion Assistant** also includes **code templates that can be used to quickly insert code fragments** into Schematron documents.

**RELAX NG/XML Schema with Embedded Schematron Rules**

Schematron rules can be embedded into an XML Schema through annotations (using the `appinfo` element), or in any element on any level of a RELAX NG Schema (taking into account that the RELAX NG validator ignores all elements that are not in the RELAX NG namespace).

Oxygen XML Developer plugin accepts such documents as Schematron validation schemas and it is able to extract and use the embedded rules. To validate an XML document with both RELAX NG schema and its embedded Schematron rules, you need to associate the document with both schemas. For example:

```xml
<?xml-model href="percent.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<?xml-model href="percent.rng" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

The second association validates your document with Schematron rules extracted from the RELAX NG Schema. Similarly, you can specify an XML Schema having the embedded Schematron rules.

```xml
<?xml-model href="percent.xsd" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

**Note:** When you work with XML Schema or Relax NG documents that have embedded Schematron rules Oxygen XML Developer plugin provides two built-in validation scenarios: Validate XML Schema with embedded Schematron for XML schema, and Validate Relax NG with embedded Schematron for Relax NG. You can use one of these scenarios to validate the embedded Schematron rules.

**Editing Schematron Schema in the Master Files Context**

Smaller interrelated modules that define a complex Schematron cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a diagnostic defined in a main schema document is not visible when you edit an included module. Oxygen XML Developer plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.
You can set a main Schematron document either using the master files support from the Navigator view, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Developer plugin warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- Correct validation of a module in the context of a larger schema structure.
- Content Completion Assistant displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one.

**Schematron Resource Hierarchy/Dependencies View**

The Resource Hierarchy/Dependencies view allows you to see the hierarchy/dependencies for a Schematron schema. To open this view, go to Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies.

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

![Image of Resource Hierarchy/Dependencies View]

**Figure 154: Resource Hierarchy/Dependencies View**

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

![Image of Resource Hierarchy/Dependencies View - Dependencies for table_abstract.sch]

**Figure 155: Resource Hierarchy/Dependencies View - Dependencies for table_abstract.sch**

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

- **Refresh**
  - Refreshes the Hierarchy/Dependencies structure.

- **Stop**
  - Stops the hierarchy/dependencies computing.

- **Show Hierarchy**
  - Allows you to choose a resource to compute the hierarchy structure.
Show Dependencies
Allows you to choose a resource to compute the dependencies structure.

Configure
Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History
Provides access to the list of previously computed dependencies. Use the **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open
Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location
Copies the location of the resource.

Move resource
Moves the selected resource.

Rename resource
Renames the selected resource.

Show Resource Hierarchy
Shows the hierarchy for the selected resource.

Show Resource Dependencies
Shows the dependencies for the selected resource.

Add to Master Files
Adds the currently selected resource in the **Master Files** directory.

Expand All
Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All
Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

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

**Note:** The **Move resource** or **Rename resource** actions give you the option to update the references to the resource.

Moving/Renaming Schematron Resources
You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
• Update references of the moved resource(s) - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

Highlight Component Occurrences in Schematron Documents

When you position your mouse cursor over a component in a Schematron document, Oxygen XML Developer plugin searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default behaviour of Highlight Component Occurrences, open the Preferences dialog box and go to Editor > Mark Occurrences. You can also trigger a search using the Search > Search Occurrences in File Ctrl Shift U (Command Shift U on OS X) action from contextual menu. Matches are displayed in separate tabs of the Results view.

Searching and Refactoring Operations in Schematron Documents

Search Actions

The following search actions can be applied on pattern, phase, or diagnostic types and are available from the Search submenu in the contextual menu of the current editor:

• Search References - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.

• Search References in... - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the Search References dialog box.

• Search Declarations - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

• Search Declarations in... - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.

• Search Occurrences in File - Searches all occurrences of the item at the caret position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on pattern, phase, or diagnostic types and are available from the Refactoring submenu in the contextual menu of the current editor:

• Rename Component - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the Esc or Enter key on your keyboard.

• Rename Component in... - Opens the Rename component_type dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the Preview button, you can view the files affected by the Rename Component action.
Searching and Refactoring Operations Scope in Schematron Documents

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the *Change scope* operation, available in the Quick Fix action set or on the Resource Hierarchy/Dependency View toolbar. You can restrict the scope to the current project or to one or multiple working sets. The *Use only Master Files, if enabled* check-box allows you to restrict the scope of the search and refactor operations to the resources from the Master Files directory. Click *read more* for details about the Master Files support.

Quick Assist Support in Schematron Documents

*Quick Assist* is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the quick assist menu if you press `Ctrl + 1` keys (Meta 1 on Mac OS X) on your keyboard.
Figure 158: Schematron Quick Assist Support

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

- **Rename Component in...**
  Renames the component and all its dependencies.

- **Search Declarations**
  Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

- **Search References**
  Searches all references of the component in a predefined scope.

- **Component Dependencies**
  Searches the component dependencies in a predefined scope.

- **Change Scope...**
  Configures the scope that will be used for future search or refactor operations.

- **Rename Component**
  Allows you to rename the current component in-place.

- **Search Occurrences**
  Searches all occurrences of the component within the current file.

**Editing Schematron Quick Fixes**

Oxygen XML Developer plugin provides support for editing the Schematron Quick Fixes. You can define a library of quick fixes by editing them directly in the current Schematron file or in a separate file. Oxygen XML Developer plugin assists you in editing Schematron Quick Fixes with schema-based content completion, syntax highlighting, and validation as you type.

This section includes details about the Schematron Quick Fixes feature and how to customize them.

**Validating Schematron Quick Fixes**

By default, Schematron Quick Fixes are validated as you edit them within the Schematron file or while editing them in a separate file. To change this, open the Preferences dialog box, go to Editor > Document Checking, and disable the Enable automatic validation option.

To validate Schematron Quick Fixes manually, select the Validate action from the Validation toolbar drop-down list or the XML menu. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

**Content Completion in SQF**

Oxygen XML Developer plugin helps you edit Schematron Quick Fixes embedded in a Schematron document, offering, through the Content Completion Assistant, items that are valid at the caret position. When you edit the value of an
attribute that refers a quick fix id, the ids are collected from the entire definition scope. For example, if the editing context is `assert/@sqf:fix`, the Content Completion Assistant proposes all fixes defined locally and globally.

If the editing context is an attribute value that is an XPath expression (such as `sqf:add/@match` or `replace/@select`), the Content Completion Assistant offers the names of XPath functions, the XPath axes, and user-defined variables and parameters.

The Content Completion Assistant displays XSLT 1.0 functions (and optionally XSLT 2.0 / 3.0 functions) in the attributes `path`, `select`, `context`, `subject`, and `test`, depending on the Schematron options that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.6.0.5 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion also displays the XSLT Saxon extension functions.

**Highlight Quick Fix Occurrences in SQF**

When you position your mouse cursor over a quick fix id in a Schematron document, Oxygen XML Developer plugin searches for the quick fix declaration and all its references and highlights them automatically.

Customizable colors are used: one for the quick fix definition and another one for its references. Occurrences are displayed until another quick fix is selected.

To change the default behaviour of Highlight Component Occurrences, open the Preferences dialog box and go to Editor > Mark Occurrences. You can also trigger a search using the Search > Search Occurrences in File (Ctrl Shift U (Command Shift U on OS X)) action from contextual menu. Matches are displayed in separate tabs of the Results view.

**Searching and Refactoring Operations in SQF**

**Search Actions**

The following search actions can be applied on quick fix ids and are available from the Search submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.

- **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the Search References dialog box.

- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.

- **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.

- **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.

**Refactoring Actions**

The following refactoring actions can be applied on quick fix ids and are available from the Refactoring submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the caret position are updated in real time to all occurrences of the component. To exit in-place editing, press the Esc or Enter key on your keyboard.

- **Rename Component in...** - Opens the Rename component_type dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the Preview button, you can view the files affected by the Rename Component action.
Embed Schematron Quick Fixes in Relax NG or XML Schema

Schematron Quick Fixes can be embedded into a Relax NG or XML Schema within the Schematron rules from annotations (using the `appinfo` element), or in any Schematron rule of a RELAX NG Schema.

Oxygen XML Developer plugin is able to extract and use the embedded Schematron Quick Fixes. To make the Schematron Quick Fixes available, validate the document with both the RELAX NG schema and its embedded Schematron rules.

Customizing Schematron Quick Fixes

You can customize Schematron Quick Fixes by editing them directly in the current Schematron file or in a separate file. The Schematron Quick Fixes are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. You can refer the quick fixes from the `assert` or `report` elements in the values of the `sqf:fix` attributes.

Defining a Schematron Quick Fix

The basics of a Schematron Quick Fix is defined by an ID, name, description, and the operations to be executed.

- **ID** - Defined by the `id` attribute from the `fix` element and must be unique in the current context. It is used to refer the quick fix from a `report` or `assert` element.
- **Name** - The name of the quick fix is defined by the `title` element.
- **Description** - Defined by the text in the paragraphs (p) of the `description` element.
- **Operations** - The following types of operations are supported:
  - `<sqf:add>` - To add a new node or fragment in the document.
  - `<sqf:delete>` - To remove a node from the document.
  - `<sqf:replace>` - To replace a node with another node or fragment.
  - `<sqf:stringReplace>` - To replace text content with other text or a fragment.
The assertion message that generates the quick fix is added as the description of the problem to be fixed. The title is presented as the name of the quick fix. The content of the paragraphs (p) within the description element are presented in the tooltip message when the quick fix is selected.

**Schematron Quick Fix Operations**

**Add**

The `<sqf:add>` element allows you to add a node to the instance. An anchor node is required to select the position for the new node. The anchor node can be selected by the match attribute. Otherwise, it is selected by the context attribute of the rule.

The target attribute defines the name of the node to be added. It is required if the value of the node-type attribute is set to anything other than "comment".

The `<sqf:add>` element has a position attribute and it determines the position relative to the anchor node. The new node could be specified as the first child of the anchor node, the last child of the anchor node, before the anchor node, or after the anchor node (first-child is the default value). If you want to add an attribute to the anchor node, do not use the position attribute.

**Note:** If you insert an element and its content is empty, Oxygen XML Developer plugin will insert the required element content.

**An Example of the `<sqf:add>` Element:**

```xml
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
  <rule context="head">
    <assert test="title" sqf:fix="addTitle">title element is missing.</assert>
    <sqf:fix id="addTitle">
      <sqf:title>Insert title element.</sqf:title>
      <sqf:add target="title" node-type="element">
        Title text
      </sqf:add>
    </sqf:fix>
  </rule>
</schema>
```

**Specific Add Operations:**

- **Insert Element** - To insert an element, use the `<sqf:add>` element, set the value of the node-type to "element", and specify the element QName with the target attribute. If the element has a prefix, it must be defined in the Schematron using a namespace declaration (<ns uri="namespace" prefix="prefix"/>).
- **Insert Attribute** - To insert an attribute, use the `<sqf:add>` element, set the value of the node-type to "attribute", and specify the attribute QName with the target attribute. If the attribute has a prefix, it must be defined in the Schematron using a namespace declaration (<ns uri="namespace" prefix="prefix"/>).
• **Insert Fragment** - If the `node-type` is not specified, the `<sqf:add>` element will insert an XML fragment. The XML fragment must be well formed. You can specify the fragment in the add element or by using the `select` attribute.

• **Insert Comment** - To insert a comment, use the `<sqf:add>` element and set the value of the `node-type` to "comment".

• **Insert Processing Instruction** - To insert a processing instruction, use the `<sqf:add>` element, set the value of the `node-type` to "pi" or "processing-instruction", and specify the name of the processing instruction in the `target` attribute.

**Delete**

The `<sqf:delete>` element allows you to remove any type of node (such as elements, attributes, text, comments, or processing instructions). To specify nodes for deletion the `<sqf:delete>` element can include a `match` attribute that is an XPath expression (the default value is `.`). If the `match` attribute is not included, it deletes the context node of the Schematron rule.

**An Example of the `<sqf:delete>` Element:**

```xml
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process">
    <pattern>
        <rule context="*[@xml:lang]">
            <report test="@xml:lang" sqf:fix="remove_lang">
                The attribute "xml:lang" is forbidden.</report>
            <sqf:fix id="remove_lang">
                <sqf:description>
                    <sqf:title>Remove "xml:lang" attribute</sqf:title>
                    <sqf:delete match="@xml:lang"/>
                </sqf:description>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

**Replace**

The `<sqf:replace>` element allows you to replace nodes. Similar to the `<sqf:delete>` element, it can include a `match` attribute. Otherwise, it replaces the context node of the rule. The `<sqf:replace>` element has three tasks. It identifies the nodes to be replaced, defines the replacing nodes, and defines their content.

**An Example of the `<sqf:replace>` Element:**

```xml
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
    <pattern>
        <rule context="title">
            <report test="exists(ph)" sqf:fix="resolvePh" role="warn">
                ph element is not allowed in title.</report>
            <sqf:fix id="resolvePh">
                <sqf:description>
                    <sqf:title>Change the ph element into text</sqf:title>
                    <sqf:replace match="ph">
                        <value-of select="."/>
                    </sqf:replace>
                </sqf:description>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

**Other Attributes for Replace Operations:**

• `node-type` - Determines the type of the replacing node. The permitted values include:
  - `keep` - Keeps the node type of the node to be replaced.
  - `element` - Replaces the node with an element.
  - `attribute` - Replaces the node with an attribute.
  - `pi` - Replaces the node with a processing instruction.
  - `comment` - Replaces the node with a comment.

• `target` - By using a *QName* it gives the replacing node a name. This is necessary when the value of the `node-type` attribute is anything other than "comment".
• **select** - Allows you to choose the content of the replacing nodes. You can use XPath expressions with the `select` attribute. If the `select` attribute is not specified then the content of the `<sqf:replace>` element is used instead.

**String Replace**

The `<sqf:stringReplace>` element is different from the others. It can be used to find a sub-string of text content and replace it with nodes or other strings.

**Attributes for the String Replace Operation:**

- **match** - Allows you to select text nodes that contain the sub-strings you want to replace.
- **select** - Allows you to select the replacing fragment, in case you do not want to set it in the content of the `stringReplace` element.
- **regex** - Matches the sub-strings using a regular expression.

**Note:** Regular expressions in the `<sqf:stringReplace>` element always has the *dot matches all* flag set to “true”. Therefore, the line terminator will also be matched by the regular expression.

**Attention:** The context of the content within the `<sqf:stringReplace>` element is set to the whole text node, rather than the current sub-string.

**An Example of the `<sqf:stringReplace>` Element:**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
            xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
  <sch:pattern>
    <sch:rule context="text()">
      <sch:report test="matches(., '[oO][xX]ygen')" sqf:fix="changeWord">
        The oXygen word is not
      </sch:report>
      <sch:fix id="changeWord" regex="[oO][xX]ygen">
        <sqf:description>
          <sqf:title>Replace word with product</sqf:title>
          <sqf:description>
            The oXygen word is not
          </sqf:description>
          <sqf:stringReplace regex="[oO][xX]ygen">
            <ph keyref="product"/>
          </sqf:stringReplace>
        </sqf:description>
      </sch:fix>
    </sch:report>
  </sch:rule>
</sch:pattern>
</sch:schema>
```

**Formatting and Indenting Inserted Content**

The content that is inserted by the **Add**, **Replace**, or **String Replace** operations is automatically indented unless you set the value of the `xml:space` attribute to `preserve` on the operation element. There are several methods available to format the content that is inserted:

- **xsl:text** - You can use an `xsl:text` element to format the inserted content and keep the automatic indentation, as in the following example:

```
<sqf:add position="last-child">
  <row>
    <xsl:text>
      <entry>First column</entry>
    </xsl:text>
    <entry>Second column</entry>
  </row>
</sqf:add>
```

- **xml:space** - Use the `xml:space` attribute and set its value to `preserve` to format the content and specify the spacing between elements, as in the following example:

```
<sqf:add node-type="element" target="codeblock" xml:space="preserve">
  /* a long sample program */
  Do forever
    Say "Hello, World"
  End
</sqf:add>
```
The *Use-When Condition*

To restrict a quick fix or a specific operation to only be available if certain conditions are met, the `use-when` attribute can be included in the `<sqf:fix>` element or any of the SQF operation elements. The condition of the `use-when` attribute is an XPath expression and the fix or operation will be performed only if the condition is satisfied. In the following example, the `use-when` condition is applied to the `<sqf:fix>` element:

```xml
<sqf:fix id="last" use-when="$colWidthSummarized - 100 lt $lastWidth" role="replace">
  <sqf:fix>
    <sqf:title>Subtract the excessive width from the last element.</sqf:title>
  </sqf:fix>
  <let name="delta" value="$colWidthSummarized - 100"/>
  <sqf:add match="html:col[last()]" target="width" node-type="attribute">
    <let name="newWidth" value="number(substring-before(@width,'%')) - $delta"/>
  </sqf:add>
</sqf:fix>
```

Additional Elements Supported in the Schematron Quick Fixes

**<sqf:call-fix>**

This element calls another quick fix within a quick fix. The called quick fix must be defined globally or in the same Schematron rule as the calling quick fix. A calling quick fix adopts the activity elements of the called quick fix and should not include other activity elements. You can also specify which parameters are sent by using the `<sqf:with-param>` child element.

**<sqf:group>**

Allows you to group multiple quick fixes and refer them from an `assert` or `report` element.

**<sqf:fixes>**

Is defined globally and contains global fixes and groups of fixes.

**<sqf:keep>**

Used to copy the selected nodes that are specified by the `select` attribute.

**Note:** In Oxygen XML Developer plugin the copied nodes cannot be manipulated by the current or other activity elements.

**<sqf:param>**

Defines a parameter for a quick fix. If the parameter is defined as `abstract` then the `type` and default value should not be specified and the fix can be called from an abstract pattern that defines this parameter.

**Warning:** The `<sqf:user-entry>` element is not supported and if it is used then the quick fix will not be displayed.

**Note:** The `sqf:defaultFix` attribute is also ignored in Oxygen XML Developer plugin.


**Editing XHTML Documents**

XHTML documents with embedded CSS, JS, PHP, and JSP scripts are rendered with dedicated coloring schemes. To customize them, open the `Preferences` dialog box and go to `Editor > Syntax Highlight`.

**Spell Checking**

The `Spelling` dialog allows you to check the spelling of the edited document. To open this dialog, click the `Check Spelling` toolbar button.
Figure 160: The Check Spelling Dialog

The dialog contains the following fields:

- **Unrecognized word** - Contains the word that cannot be found in the selected dictionary. The word is also highlighted in the XML document.
- **Replace with** - The character string which is suggested to replace the unrecognized word.
- **Guess** - Displays a list of words suggested to replace the unknown word. Double click a word to automatically insert it in the document and resume the spell checking process.
- **Default language** - Allows you to select the default dictionary used by the spelling engine.
- **Paragraph language** - In an XML document you can mix content written in different languages. To tell the spell checker engine what language was used to write a specific section, you need to set the language code in the `lang` or `xml:lang` attribute to that section. Oxygen XML Developer plugin automatically detects such sections and instructs the spell checker engine to apply the appropriate dictionary.
- **Replace** - Replaces the currently highlighted word in the XML document, with the selected word in the **Replace with** field.
- **Replace All** - Replaces all occurrences of the currently highlighted word in the XML document, with the selected word in the **Replace with** field.
- **Ignore** - Ignores the first occurrence of the unrecognized word and allows you to continue checking the document. Oxygen XML Developer plugin skips the content of the XML elements *marked as ignorable*.
- **Ignore All** - Ignores all instances of the unknown word in the current document.
- **Learn** - Includes the unrecognized word in the list of valid words.
- **Options** - Sets the configuration options of the spell checker.
- **Begin at caret position** - Instructs the spell checker to begin checking the document starting from the current cursor position.
- **Close** - Closes the dialog.

**Spell Checking Dictionaries**

There are two spell checking engines available in Oxygen XML Developer plugin: **Hunspell** checker (default setting) and **Java** checker. You can set the spell check engine in the **Spell checking engine** preferences page. The dictionaries
used by the two engines differ in format, so you need to follow specific procedures in order to add another dictionary to your installation of Oxygen XML Developer plugin.

**Dictionaries for the Hunspell Checker**

The Hunspell spell checker is open source and has LGPL license. The format of the Hunspell spell dictionary is supported by Mozilla, OpenOffice and the Chrome browser. Oxygen XML Developer plugin comes with the following built-in dictionaries for the Hunspell checker:

- English (US)
- English (UK)
- French
- German
- Spanish.

Each language-country variant combination has its specific dictionary. If you cannot find a Hunspell dictionary that is already built for your language, you can build the dictionary you need. To build a dictionary from this list follow these instructions.

**Add Dictionaries and Term Lists for the Hunspell Checker**

To add new spelling dictionaries to Oxygen XML Developer plugin, or to replace an existing one, follow these steps:

1. Download the files you need for your language dictionary.
2. The downloaded .oxt file is a zip archive. If you are creating a new dictionary, copy the .aff and .dic files from this archive in the spell subfolder of the Oxygen XML Developer plugin preferences folder. The Oxygen XML Developer plugin preferences folder is $[APPLICATION-DATA-FOLDER]$, where $[APPLICATION-DATA-FOLDER]$ is:
   - C:\Users\[LOGIN-USER-NAME]\AppData\Roaming on Windows Vista, Windows 7, and Windows 8
   - [USER-HOME-FOLDER]/Library/Preferences on OS X
   - [USER-HOME-FOLDER] on Linux
3. If you are updating an existing dictionary, copy the .aff and .dic files into the folder $[OXYGEN_DIR]/dicts/spell$.
4. Restart the application after copying the dictionary files.

**Note:** You can setup Oxygen XML Developer plugin to use dictionaries and term lists from a custom location configured in the Dictionaries preferences page.

**Dictionaries for the Java Checker**

A Java spell checker dictionary has the form of a .dar file located in the directory $[OXYGEN_DIR]/dicts$. Oxygen XML Developer plugin comes with the following built-in dictionaries for the Java checker:

- English (US)
- English (UK)
- English (Canada)
- French (France)
- French (Belgium)
- French (Canada)
- French (Switzerland)
- German (old orthography)
- German (new orthography)
- Spanish

A pre-built dictionary can be added by copying the corresponding .dar file to the folder $[OXYGEN_DIR]/dicts$ and restarting Oxygen XML Developer plugin. There is one dictionary for each language-country variant combination.
Learned Words

Spell checker engines rely on dictionary to decide that a word is correctly spelled. To tell the spell checker engine that an unknown word is actually correctly spelled, you need to add that word to its dictionary. There are two ways to do this:

- Press the Learn button from the Spelling dialog box.
- Invoke the contextual menu on an unknown word, then press Learn word.

Learned words are stored into a persistent dictionary file. Its name is composed of the currently checked language code and the .tdi extension, for example en_US.tdi. It is located in the:

- [HOME_DIR]/Application Data/com.oxygenxml.developer/spell folder on Windows XP.
- [HOME_DIR]/Library/Preferences/com.oxygenxml.developer/spell folder on Mac OS X.
- [HOME_DIR]/com.oxygenxml.developer/spell folder on Linux.

Note: To change this folder go to the Editor > Spell Check > Dictionaries preferences page.

Note: To delete items from the list of learned words, press Delete learned words in the Editor > Spell Check > Dictionaries preferences page.

Ignored Words

The content of some XML elements like programlisting, codeblock or screen should always be skipped by the spell checking process. The skipping can be done manually word by word by the user using the Ignore button of the Spelling dialog or, more conveniently, automatically by maintaining a set of known element names that should never be checked. You maintain this set of element names in the user preferences as a list of XPath expressions that match the elements.

Only a small subset of XPath expressions is supported, that is only the '/' and '// separators and the '*' wildcard. Two examples of supported expressions are /a/*/b and //c/d/*.

Automatic Spell Check

To allow Oxygen XML Developer plugin to automatically check the spelling as you write, you need to enable the Automatic spell check option from the Spell Check preferences page. Unknown words are highlighted and feature a contextual menu which offers the following actions:

Delete Repeated Word
- Allows you to delete repeated words.

Learn Word
- Allows you to add the current unknown word to the persistent dictionary.

Spell check options
- Opens the Spell Check preferences page.

Also, a list of words suggested by the spell checking engine as possible replacements of the unknown word is offered in the contextual menu.

Spell Checking in Multiple Files

The Check Spelling in Files action allows you to check the spelling on multiple local or remote documents. This action is available in:

- The contextual menu of the Navigator view.

The spelling corrections are displayed in the Results view, that allows you to group the reported errors as a tree with two levels.
The following scopes are available:

- **All opened files** - The spell check is performed in all opened files.
- **Directory of the current file** - All the files in the folder of the current edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - Checks the spelling in the files located at a path that you specify.

The **Options** section includes the following options:

- **File filter** - Allow you to filter the files from the selected scope.
- **Recurse subdirectories** - When enabled, the spell check is performed recursively for the specified scope. The one exception is that this option is ignored if the scope is set to **All opened files**.
- **Include hidden files** - When enabled, the spell check is also performed in the hidden files.
- **Spell Check Options** - The spell check processor uses the options available in the *Spell Check preferences panel*.

### Handling Read-Only Files

The default workbench behavior applies when editing read-only files in the **Text** mode. For all other modes no modification is allowed as long as the file remains read-only.

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

### Associating a File Extension with Oxygen XML Developer plugin

To associate a file extension with Oxygen XML Developer plugin on Windows:

- Go to the **Start** menu and click **Control Panel**.
- Go to **Default Programs**.
- Click **Associate a file type or protocol with a program**.
- Click the file extension you want to associate with Oxygen XML Developer plugin, then click **Change program**.
- In the **Open With** dialog box, click **Browse** and navigate to Oxygen XML Developer plugin.
To associate a file extension with Oxygen XML Developer plugin on Mac:

- In **Finder**, right click a file and from the contextual menu select **Get Info**.
- In the **Open With** subsection, select **Other** from the application combo and browse to Oxygen XML Developer plugin.
- With Oxygen XML Developer plugin selected, click **Change All**.
Chapter 6

Predefined Document Types

Topics:

• **Document Type**

  The following pre-defined document types are supported in Oxygen XML Developer plugin and each of these document types include built-in transformation scenarios and document templates:

• **DocBook 4** - A document type standard for books, articles, and other prose documents (particularly technical documentation).
• **DocBook 5** - An enhanced (version 5) document type standard designed for a variety of documents (particularly technical documentation).
• **DITA** - An XML-based architecture designed for authoring, producing, and delivering technical information.
• **DITA Map** - A document type that collects and organizes references to DITA topics or other maps.
• **XHTML** - Extensible HyperText Markup Language includes the same depth of expression as HTML, but also conforms to XML syntax.
• **TEI ODD** - Text Encoding Initiative One Document Does it all is an XML-conformant specification that allows you to create TEI P5 schema in a literate programming style.
• **TEI P4** - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
• **TEI P5** - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
• **JATS** - The NISO Journal Article Tag Suite is a technical standard that defines an XML format for scientific literature.

Oxygen XML Developer plugin also provides limited support and includes document templates for a variety of other document types, including:

• **EPUB (NCX, OCF, OPF 2.0 & 3.0)** - A standard for e-book files.
• **DocBook Targetset** - For resolving cross-references when using *olinks*.
• **XSLT Stylesheets** - A document type that provides a visual mode for editing XSLT stylesheets.
• **WSDL** - Web Services Description Language is an XML language for describing the functionality offered by a web service.
• **Schematron** - For making assertions about the presence or absence of patterns in XML documents. This document type applies to the ISO Schematron version.
• **Schematron Quick Fixes (SQF)** - An extension of the ISO standard Schematron, allows developers to define *Quick Fixes* for Schematron errors.
• **StratML (Part 1 & 2)** - Part 1 and 2 of the Strategy Markup Language specification.
• **XProc** - A document type for processing XProc script files.
• **XML Schema** - Documents that provide support for editing annotations.
• XML Spec - A markup language for W3C specifications and other technical reports.
• DITAVAL - DITA conditional processing profile to identify the values you want to conditionally process for a particular output, build, or other purpose.
• Daisy - A technical standard for digital audio books, periodicals, and computerized text. It is designed to be an audio substitute for print material.
• EAD - Encoded Archival Description is an XML standard for encoding archival finding aids.
• KML - Keyhole Markup Language is an XML notation for expressing geographic visualization in maps and browsers.
• Maven Project & Settings - Project or settings file for Maven build automation tool that is primarily used for Java projects.
• Oasis XML Catalog - A document that describes a mapping between external entity references and locally-cached equivalents.
• XLIFF (1.2 & 2.0) - XML Localization Interchange File Format is a standard for passing data between tools during a localization process.
A document type or framework is associated to an XML file according to a set of rules. It also includes a variety of settings that improve editing capabilities in the Author mode for its particular file type. These settings include:

- A default grammar used for validation and content completion in both Author mode and Text mode.
- CSS stylesheets for rendering XML documents in Author mode.
- User actions invoked from toolbar or menu actions in Author mode.
- Predefined scenarios used for transformations for the class of XML documents defined by the document type.
- XML catalogs.
- Directories with file templates.
- User-defined extensions for customizing the interaction with the content author in Author mode.

Oxygen XML Developer plugin comes with built-in support for many common document types. Each document type is defined in a framework. You can create new frameworks or make changes to existing frameworks to suit your individual requirements.

To see a video on configuring a framework in Oxygen XML Developer plugin, go to http://oxygenxml.com/demo/FrameworkConfiguration.html.

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a DocBook 4 document when one of the following conditions are true:

- The root element name is book or article.
- The PUBLIC ID of the document contains the string 
  -//OASIS//DTD DocBook XML.

The default schema, docbookx.dtd, for these documents is stored in

[OXYGEN_DIR]/frameworks/docbook/4.5/dtd/.

The default XML catalog, catalog.xml, is stored in [OXYGEN_DIR]/frameworks/docbook/.

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, EPUB and EPUB 3.

WebHelp Output

DocBook 4 documents can be transformed into WebHelp systems, such as:

1. Click Configure Transformation Scenarios.
3. Click Apply associated.

When the DocBook WebHelp transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- use.stemming - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).
- webhelp.copyright - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
• **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```html
<div id="facebook">
  "<script>
<!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = '//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0'; fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
</div>
```

• **webhelp.footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.

• **l10n:gentext.default.language** - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.

• **webhelp.logo.image** - Specifies a path to an image displayed as a logo in the left side of the output header.

• **webhelp.logo.image.target.url** - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

• **webhelp.search.ranking** - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).

### WebHelp With Feedback Output

To publish DocBook 4 to WebHelp With Feedback, follow these steps:

1. Click **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp with Feedback** scenario from the **DocBook 4** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

When the **DocBook WebHelp with Feedback** transformation is complete, your default browser opens the **installation.html** file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To further customize the out-of-the-box transformation, you can edit its parameters:

• **use.stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

• **webhelp.copyright** - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).

• **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```html
<div id="facebook">
  "<script>
<!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = '//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0'; fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
</div>
```
• webhelp.footer.include - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the webhelp.footer.file parameter has a value, then the content of that file is used as footer. If the webhelp.footer.file has no value then the default Oxygen footer is inserted in each WebHelp page.

• l10n.gentext.default.language - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is en or for French it is fr, and so on.

• webhelp.logo.image - Specifies a path to an image displayed as a logo in the left side of the output header.

• webhelp.logo.image.target.url - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

• webhelp.search.ranking - If this parameter is set to false then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is true).

• webhelp.product.id - This parameter specifies a short name for the documentation target, or product (for example, mobile-phone-user-guide, hvac-installation-guide). You can deploy documentation for multiple products on the same server.

Note: The following characters are not allowed in the value of this parameter: < > / \ ' ( ) { }  

• webhelp.product.version - This parameter specifies the documentation version. New comments are bound to this version. Multiple documentation versions can be deployed on the same server.

Note: The following characters are not allowed in the value of this parameter: < > / \ ' ( ) { }  


To watch our video demonstration about the feedback-enabled WebHelp system, go to http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Mobile Output
To generate a mobile WebHelp system from your DocBook 4 document, follow these steps:

1. Click Configure Transformation Scenarios.
3. Click Apply associated.

To further customize the out-of-the-box transformation, you can edit its parameters:

• use.stemming - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).
• webhelp.copyright - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
• l10n.gentext.default.language - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is en or for French it is fr, and so on.
• webhelp.footer.file - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate div or span element, and the code for each script element is included in an XML comment (also, the start and end
tags for the XML comment must be on a separate line). The following code excerpt is an example for adding a Facebook™ widget:

```html
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0"; fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk'); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.

When the DocBook WebHelp - Mobile transformation is complete, the output is automatically opened in your default browser.

### DocBook to PDF Output Customization

Main steps for customization of PDF output generated from DocBook XML documents.

When the default layout and output look of the DocBook to PDF transformation need to be customized, the following main steps should be followed. In this example a company logo image is added to the front matter of a book. Other types of customizations should follow some similar steps.


You should start from a copy of the file `[OXYGEN_DIR]/frameworks/docbook/xsl/fo/titlepage.templates.xml` and customize it. The instructions for the spec file can be found [here](#).

An example of spec file:

```xml
<titlepage-content t:side="recto">
  <mediaobject/>
  <title>
    <named-template t:book.verso.title font-size="#size2;" font-weight="bold" font-family="{$title.font.family}"/>
    <corpauthor/>
    ...
  </titlepage-content>
```

2. Generate a new XSLT stylesheet from the title spec file from the previous step.

Apply `[OXYGEN_DIR]/frameworks/docbook/stylesheet/template/titlepage.xsl` to the title spec file. The result is an XSLT stylesheet, let's call it `mytitlepages.xsl`.

3. Import `mytitlepages.xsl` in a DocBook customization layer.

The customization layer is the stylesheet that will be applied to the XML document. The `mytitlepages.xsl` should be imported with an element like:

```xml
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert logo image in the XML document.

The path to the logo image must be inserted in the `book/info/mediaobject` element of the XML document.

5. Apply the customization layer to the XML document.

A quick way is duplicating the transformation scenario DocBook PDF that comes with Oxygen and set the customization layer in the XSL URL property of the scenario.
DocBook to EPUB Transformation

The EPUB specification recommends the use of OpenType fonts (recognized by their .otf file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:

```css
@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:

```css
h1 {
  font-size: 20pt;
  margin-bottom: 20px;
  font-weight: bold;
  font-family: "MyFont";
  text-align: center;
}
```

- in your DocBook to EPUB transformation, set the epub.embedded.fonts parameter to fonts/MyFont.otf. If you need to provide more files, use comma to separate their file paths.

Note: The html.stylesheet parameter allows you to include a custom CSS in the output EPUB.

DocBook 4 Templates

Default templates are available in the New File wizard. You and can use them to create a skeletal form of a DocBook 4 book or article. These templates are stored in the [OXYGEN_DIR]/frameworks/docbook/templates/DocBook 4 folder.

Here are some of the DocBook 4 templates available when creating new documents from templates.

- Article
- Article with MathML
- Article with SVG
- Article with XInclude
- Book
- Book with XInclude
- Chapter
- Section
- Set of Books

Inserting olink Links in DocBook 5 Documents

An olink is a type of link between two DocBook XML documents.

The olink element is the equivalent for linking outside the current DocBook document. It has the attribute targetdoc for the document ID that contains the target element and the attribute targetptr for the ID (the value of an id or xml:id attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the Administrator Guide is a book with the document ID MailAdminGuide and it contains a chapter about user accounts like the following:

```xml
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...
```

You can form a cross reference to that chapter by adding an olink in the User Guide like the following:

```xml
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
```

You may need to update your
when you get a new machine.

1. Decide what documents are included in the domain for cross referencing.

   An ID should be assigned to each document that will be referenced with an olink. Usually it is added as an id or xml:id attribute to the root element of the document. A document ID is a string that is unique for each document in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like ug, ag, and ref or more specific IDs like MailUserGuide, MailAdminGuide, and MailReference.

2. Decide the output hierarchy.

   For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element `<dir name="directory_name">` in the target database document. In the example from the next step the hierarchy is documentation/guides/mailuser, documentation/guides/mailadmin, documentation/guides/reference.

3. Create the target database document.

   Each collection of documents has a master target database document that is used to resolve all olinks from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically.

   An example is the following:

   ```xml
   <?xml version="1.0" encoding="utf-8"?>
   <!DOCTYPE targetset [ 
   <!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db"> 
   <!ENTITY agtargets SYSTEM "file:///doc/adminguide/target.db"> 
   <!ENTITY reftargets SYSTEM "file:///doc/man/target.db"> ]>
   <targetset>
   <targetsetinfo>
   Description of this target database document, which is for the examples in olink doc.
   </targetsetinfo>
   <!-- Site map for generating relative paths between documents -->
   <sitemap>
   <dir name="guides">
   <document targetdoc="MailUserGuide" bas-uri="userguide.html">
   &ugtargets;
   </document>
   </dir>
   <dir name="mailadmin">
   <document targetdoc="MailAdminGuide">
   &agtargets;
   </document>
   </dir>
   <dir name="reference">
   <document targetdoc="MailReference">
   &reftargets;
   </document>
   </dir>
   </sitemap>
   </targetset>
   
   An example of a target.db file:

   ```xml
   <div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">
   <ttl>Administering User Accounts</ttl>
   <xreftext>How to administer user accounts</xreftext>
   <div element="part" href="#d5e4" number="1">
   <ttl>First Part</ttl>
   <xreftext>Part I, "First Part"</xreftext>
   ```
4. Generate the target data files.

These files are the target.db files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called collect.xref.targets must be set to the value yes. The default name of a target data file is target.db but it can be changed by setting an absolute file path in the XSLT parameter targets.filename.

5. Insert olink elements in the DocBook XML documents.

When a DocBook XML document is edited in Author mode Oxygen XML Developer plugin provides the Insert OLink action on the toolbar. This action allows selecting the target of an olink from the list of all possible targets from a specified target database document. In the following image the target database document is called target.xml.

![Figure 162: Insert OLink Dialog Box](image)


That is done using a DocBook transformation scenario in which the URL of the target database document is set in the target.database.document parameter. The DocBook XSL stylesheets know how to resolve olinks in the output files using the value of this parameter.
A file is considered to be a DocBook 5 document when the namespace is http://docbook.org/ns/docbook.

The default Relax NG and Schematron schema, docbookxi.rng, for these documents is stored in [OXYGEN_DIR]/frameworks/docbook/5.0/rng/.

The default XML catalog, catalog.xml, is stored in [OXYGEN_DIR]/frameworks/docbook/5.0/.

**DocBook 5 Transformation Scenarios**

Default transformation scenarios allow you to transform DocBook 5 documents to WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, EPUB, and EPUB 3.

**WebHelp Output**

DocBook 5 documents can be transformed into WebHelp systems, such as:

**WebHelp Output**

To publish DocBook 5 to WebHelp, follow these steps:

1. Click ➤ Configure Transformation Scenarios.
2. Select the DocBook WebHelp scenario from the DocBook 5 section.
3. Click Apply associated.

When the DocBook WebHelp transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- use.stemming - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).
- webhelp.copyright - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- webhelp.footer.file - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate div or span element, and the code for each script element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code excerpt is an example for adding a Facebook™ widget:

```xml
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = 'http://connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0'; js.src = 'http://www.facebook.com/social_plugin.js'; fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk'); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- webhelp.footer.include - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the webhelp.footer.file parameter has a value, then the content of that file is used as footer. If the webhelp.footer.file has no value then the default Oxygen footer is inserted in each WebHelp page.
- l10n.gentext.default.language - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is en or for French it is fr, and so on.
- webhelp.logo.image - Specifies a path to an image displayed as a logo in the left side of the output header.
- webhelp.logo.image.target.url - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- webhelp.search.ranking - If this parameter is set to false then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is true).
WebHelp With Feedback Output

To publish DocBook 5 to WebHelp With Feedback, follow these steps:

1. Click Configure Transformation Scenarios.
3. Click Apply associated.
4. Enter the documentation product ID and the documentation version.

When the DocBook WebHelp with Feedback transformation is complete, your default browser opens the installation.html file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To further customize the out-of-the-box transformation, you can edit its parameters:

- **use.stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).
- **webhelp.copyright** - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate div or span element, and the code for each script element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```xml
<div id="facebook">
  <div id="fb-root"/>
  <script>
    -- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0"; fjs.parentNode.insertBefore(js, fjs); })(document, 'script', 'facebook-jssdk'); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- **webhelp.footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the webhelp.footer.file parameter has a value, then the content of that file is used as footer. If the webhelp.footer.file has no value then the default Oxygen footer is inserted in each WebHelp page.
- **l10n.gentext.default.language** - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is en or for French it is fr, and so on.
- **webhelp.logo.image** - Specifies a path to an image displayed as a logo in the left side of the output header.
- **webhelp.logo.image.target.url** - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- **webhelp.search.ranking** - If this parameter is set to false then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is true).
- **webhelp.product.id** - This parameter specifies a short name for the documentation target, or product (for example, mobile-phone-user-guide, hvac-installation-guide). You can deploy documentation for multiple products on the same server.

**Note:** The following characters are not allowed in the value of this parameter: < > / \ ' ( ) { } = ; * % + &

- **webhelp.product.version** - This parameter specifies the documentation version. New comments are bound to this version. Multiple documentation versions can be deployed on the same server.

**Note:** The following characters are not allowed in the value of this parameter: < > / \ ' ( ) { } = ; * % + &/>
WebHelp Mobile Output

To generate a mobile WebHelp system from your DocBook 5 document, follow these steps:

1. Click Configure Transformation Scenarios.
2. Select the DocBook WebHelp - Mobile scenario from the DocBook 5 section.
3. Click Apply associated.

To further customize the out-of-the-box transformation, you can edit its parameters:

- **use.stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).
- **webhelp.copyright** - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).
- **l10n.gentext.default.language** - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is en or for French it is fr, and so on.
- **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate div or span element, and the code for each script element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```xml
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0"; js.parentNode.insertBefore(js, fjs); })('script', 'facebook-jssdk'); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

- **webhelp.footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the webhelp.footer.file parameter has a value, then the content of that file is used as footer. If the webhelp.footer.file has no value then the default Oxygen footer is inserted in each WebHelp page.

When the DocBook WebHelp - Mobile transformation is complete, the output is automatically opened in your default browser.

DocBook to PDF Output Customization

Main steps for customization of PDF output generated from DocBook XML documents.

When the default layout and output look of the DocBook to PDF transformation need to be customized, the following main steps should be followed. In this example a company logo image is added to the front matter of a book. Other types of customizations should follow some similar steps.


You should start from a copy of the file

```
[OXYGEN_DIR]/frameworks/docbook/xsl/fo/titlepage.templates.xml
```

and customize it. The instructions for the spec file can be found here.
An example of spec file:

```xml
    <t:titlepage-content t:side="recto">
        <mediaobject/>
        <title
            t:named-template="book.verso.title"
            font-size="&hsize2;"
            font-weight="bold"
            font-family="{$title.font.family}"/>
        <corpauthor/>
        ...
    </t:titlepage-content>
```

2. Generate a new XSLT stylesheet from the title spec file from the previous step. Apply `[OXYGEN_DIR]/frameworks/docbook/xsl/template/titlepage.xsl` to the title spec file. The result is an XSLT stylesheet, let's call it `mytitlepages.xsl`.

3. Import `mytitlepages.xsl` in a DocBook customization layer.
   The customization layer is the stylesheet that will be applied to the XML document. The `mytitlepages.xsl` should be imported with an element like:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert logo image in the XML document.
   The path to the logo image must be inserted in the `book/info/mediaobject` element of the XML document.

5. Apply the customization layer to the XML document.
   A quick way is duplicating the transformation scenario **DocBook PDF** that comes with Oxygen and set the customization layer in the `XSL URL property of the scenario`.

**DocBook to EPUB Transformation**

The EPUB specification recommends the use of OpenType fonts (recognized by their `.otf` file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:

```css
    @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:

```css
    h1 { 
        font-size:20pt;
        margin-bottom:20px;
        font-weight: bold;
        font-family: "MyFont";
        text-align: center;
    }
```

- in your DocBook to EPUB transformation, set the `epub.embedded.fonts` parameter to `fonts/MyFont.otf`. If you need to provide more files, use comma to separate their file paths.

   **Note:** The `html.stylesheet` parameter allows you to include a custom CSS in the output EPUB.

**DocBook 5 Templates**

Default templates are available in the **New File wizard** and can be used for easily creating a skeletal form of a DocBook 5 book or article. These templates are stored in the `[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 5` folder.

Here are some of the DocBook 5 templates available when creating **new documents from templates**.
Inserting olink Links in DocBook 5 Documents

An olink is a type of link between two DocBook XML documents.

The olink element is the equivalent for linking outside the current DocBook document. It has the attribute targetdoc for the document ID that contains the target element and the attribute targetptr for the ID (the value of an id or xml:id attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the Administrator Guide is a book with the document ID MailAdminGuide and it contains a chapter about user accounts like the following:

```
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...</title>
```

You can form a cross reference to that chapter by adding an olink in the User Guide like the following:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

1. Decide what documents are included in the domain for cross referencing.

   An ID should be assigned to each document that will be referenced with an olink. Usually it is added as an id or xml:id attribute to the root element of the document. A document ID is a string that is unique for each document in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like ug, ag, and ref or more specific IDs like MailUserGuide, MailAdminGuide, and MailReference.

2. Decide the output hierarchy.

   For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element `<dir name="directory_name">` in the target database document. In the example from the next step the hierarchy is documentation/guides/mailuser, documentation/guides/mailadmin, documentation/guides/reference.

3. Create the target database document.

   Each collection of documents has a master target database document that is used to resolve all olinks from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically. An example is the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE targetset [ 
<!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
<!ENTITY agtargets SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
<targetsetinfo>
Description of this target database document, which is for the examples in olink doc.
```
An example of a target.db file:

```xml
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">
  <ttl>Administering User Accounts</ttl>
  <xreftext>How to administer user accounts</xreftext>
  <div element="part" href="#d5e4" number="1">
    <ttl>First Part</ttl>
    <xreftext>Part I, "First Part"</xreftext>
    <div element="chapter" href="#d5e6" number="1">
      <ttl>Chapter Title</ttl>
      <xreftext>Chapter 1, Chapter Title</xreftext>
      <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
        <ttl>Section1 Title</ttl>
        <xreftext>xreflabel_here</xreftext>
      </div>
    </div>
  </div>
</div>
```

4. Generate the target data files.
   These files are the target.db files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called `collect.xref.targets` must be set to the value `yes`. The default name of a target data file is `target.db` but it can be changed by setting an absolute file path in the XSLT parameter `targets.filename`.

5. Insert `olink` elements in the DocBook XML documents.
   When a DocBook XML document is edited in Author mode, provides the Insert OLink action on the toolbar. This action allows selecting the target of an `olink` from the list of all possible targets from a specified target database document. In the following image the target database document is called `target.xml`. 

That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve olinks in the output files using the value of this parameter.

**The DITA Topics Document Type**

The Darwin Information Typing Architecture (DITA) is an XML-based architecture oriented to authoring, producing, and delivering technical information. It divides content into small, self-contained topics that you can reuse in various deliverables. The extensibility of DITA permits organizations to define specific information structures while still using standard tools to work with them. Oxygen XML Developer plugin provides schema-driven (DTD, RNG, XSD) templates for DITA documents.

A file is considered to be a DITA topic document when one of the following conditions are true:

- The root element name is one of the following: `concept`, `task`, `reference`, `dita`, or `topic`.
- The PUBLIC ID of the document is a PUBLIC ID for the elements listed above.
- The root element of the file has an attribute named `DITAArchVersion` for the “http://dita.oasis-open.org/architecture/2005/” namespace. This enhanced case of matching is only applied when the Enable DTD processing option is enabled from the Document Type Association preferences page.

The default schemas used for DITA topic documents are stored in

```
[OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/ or
[OXYGEN_DIR]/frameworks/dita/DITA-OT/schema/.
```

The default catalogs for the DITA topic document type are as follows:

- [OXYGEN_DIR]/frameworks/dita/catalog.xml
- [OXYGEN_DIR]/frameworks/dita/DITA-OT/catalog-dita.xml
- [OXYGEN_DIR]/frameworks/dita/plugin/catalog.xml
- [OXYGEN_DIR]/frameworks/dita/styleguide/catalog.xml
**DITA Transformation Scenarios**

The following default transformation scenarios are available for DITA Topics:

- **DITA XHTML** - Transforms a DITA topic to XHTML using DITA Open Toolkit.
- **DITA PDF** - Transforms a DITA topic to PDF using the DITA Open Toolkit and the Apache FOP engine.

**DITA Templates**

The default templates available for DITA topics are stored in [OXYGEN_DIR]/frameworks/dita/templates/topic folder. They can be used for easily creating a DITA concept, reference, task or topic.

Here are some of the DITA templates available when creating new documents from templates:

- **Composite** - New DITA Composite
- **Composite with MathML** - New DITA Composite with MathML
- **Concept** - New DITA Concept
- **General Task** - New DITA Task
- **Glossentry** - New DITA Glossentry
- **Glossgroup** - New DITA Glossgroup
- **Machinery Task** - New DITA Machinery Task
- **Reference** - New DITA Reference
- **Task** - New DITA Task
- **Topic** - New DITA Topic
- **Learning Assessment** - New DITA Learning Assessment (learning specialization in DITA 1.2)
- **Learning Content** - New DITA Learning Content (learning specialization in DITA 1.2)
- **Learning Summary** - New DITA Learning Summary (learning specialization in DITA 1.2)
- **Learning Overview** - New DITA Learning Overview (learning specialization in DITA 1.2)
- **Learning Plan** - New DITA Learning Plan (learning specialization in DITA 1.2)
- **Troubleshooting** - Experimental DITA 1.3 troubleshooting specialization

DITA for Publishers topic specialization templates:

- **D4P Article** - New DITA for Publishers article
- **D4P Chapter** - New DITA for Publishers chapter
- **D4P Concept** - New DITA for Publishers concept
- **D4P Conversion Configuration** - New DITA for Publishers conversion configuration
- **D4P Cover** - New DITA for Publishers cover
- **D4P Part** - New DITA for Publishers part
- **D4P Sidebar** - New DITA for Publishers sidebar
- **D4P Subsection** - New DITA for Publishers subsection
- **D4P Topic** - New DITA for Publishers topic

**The DITA Map Document Type**

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships among the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests for DITA projects.

Maps allow scalable reuse of content across multiple contexts. They can be used by information architects, authors, and publishers to plan, develop, and deliver content.

A file is considered to be a DITA map document when either of the following is true:

- The root element name is one of the following: map, bookmap.
- The public id of the document is -//OASIS//DTD DITA Map or -//OASIS//DTD DITA BookMap.
The root element of the file has an attribute named class which contains the value map/map and a DITAArchVersion attribute from the http://dita.oasis-open.org/architecture/2005/ namespace. This enhanced case of matching is only applied when the Enable DTD processing option from the Document Type Detection option page is enabled.

The default schemas used for DITA map documents are stored in [OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/ or [OXYGEN_DIR]/frameworks/dita/DITA-OT/schema/.

The default catalogs for the DITA map document type are as follows:

- [OXYGEN_DIR]/frameworks/dita/catalog.xml
- [OXYGEN_DIR]/frameworks/dita/DITA-OT/catalog-dita.xml

### DITA Map Transformation Scenarios

The following default transformations are available:

- Predefined transformation scenarios allow you to transform a DITA Map to PDF, ODF, XHTML, WebHelp, EPUB, and CHM files.
- **Run DITA-OT Integrator** - Use this transformation scenario if you want to integrate a DITA-OT plugin. This scenario runs an ANT task that integrates all the plug-ins from the DITA-OT/plugins directory.
- **DITA Map Metrics Report** - Use this transformation scenario if you want to generate a DITA Map statistics report containing information such as:
  - the number of processed maps and topics
  - content reuse percentage
  - number of elements, attributes, words, and characters used in the entire DITA Map structure
  - DITA conditional processing attributes used in the DITA Maps
  - words count
  - information types such as number of containing maps, bookmaps, or topics

Many more output formats are available by clicking the New button. The transformation process relies on the DITA Open Toolkit.

### WebHelp Output

DITA Maps can be transformed into WebHelp systems, such as:

**WebHelp Output**

To publish a DITA Map to WebHelp:

1. Click **Configure Transformation Scenarios**.
2. Select the **DITA Map WebHelp** scenario from the **DITA Map** section.
3. Click **Apply associated**.

When the **DITA Map WebHelp** transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

- **use.stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).
- **clean.output** - Deletes all files from the output folder before the transformation is performed (only no and yes values are valid and the default value is no).
- **webhelp.copyright** - Adds a small copyright text that appears at the end of the Table of Contents.
- **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate div or span element, and
the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```html
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
</div>
```

- `webhelp.footer.include` - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen footer is inserted in each WebHelp page.
- `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
- `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.
- `webhelp.search.ranking` - If this parameter is set to `false` then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is `true`).
- `args.default.language` - If the language is not detected in the DITA map, this parameter is used. The default sample value is `en-us`.
- `webhelp.search.japanese.dictionary` - The file path of the user dictionary that will be used by the Kuromoji morphological indexer that is used for indexing Japanese content in the WebHelp pages.

### WebHelp With Feedback Output

To publish a DITA Map as WebHelp with Feedback:

1. Click ⚙ Configure Transformation Scenarios.
2. Select the DITA Map WebHelp with Feedback scenario from the DITA Map section.
3. Click Apply associated.
4. Enter the documentation product ID and the documentation version.

When the DITA Map WebHelp with Feedback transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To further customize the out-of-the-box transformation, you can edit its parameters:

- `use.stemming` - Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).
- `clean.output` - Deletes all files from the output folder before the transformation is performed (only `no` and `yes` values are valid and the default value is `no`).
- `webhelp.copyright` - Adds a small copyright text that appears at the end of the Table of Contents.
- `webhelp.footer.file` - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```html
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id)) return; js = d.createElement(s); js.id = id; js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
  </script>
  <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

• **webhelp.footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the webhelp.footer.file parameter has a value, then the content of that file is used as footer. If the webhelp.footer.file has no value then the default Oxygen footer is inserted in each WebHelp page.

• **webhelp.logo.image** - Specifies a path to an image displayed as a logo in the left side of the output header.

• **webhelp.logo.image.target.url** - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

• **webhelp.search.ranking** - If this parameter is set to false then the relevance stars are no longer included in the search results displayed on the Search tab (default setting is true).

• **args.default.language** - If the language is not detected in the DITA map, this parameter is used. The default sample value is en-us.

• **webhelp.search.japanese.dictionary** - The file path of the user dictionary that will be used by the Kuromoji morphological indexer that is used for indexing Japanese content in the WebHelp pages.

To watch our video demonstration about the feedback-enabled WebHelp system, go to [http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html](http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html).

### WebHelp Mobile Output

To generate a mobile WebHelp system from your DITA Map:

1. From the DITA Maps Manager view click Configure Transformation Scenarios.
2. Select the DITA Map WebHelp - Mobile transformation scenario.
3. Click Apply associated.

When the DITA Map WebHelp - Mobile transformation is complete, the output is automatically opened in your default browser.

To further customize the out-of-the-box transformation, you can edit its parameters:

• **use.stemming** - Controls whether or not you want to include stemming search algorithms into the published output (default setting is false).

• **webhelp.copyright** - This parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output).

• **webhelp.indexer.language** - This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is en or for French it is fr, and so on.

• **webhelp.footer.file** - You can specify the path to a XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, or Google+™). The file must be well-formed, each widget must be in separate div or span element, and the code for each script element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```xml
<div id="facebook">
  <div id="fb-root"/>
</div>
<script>
  <!-- (function(d, s, id) 
    var js, fjs = d.getElementsByTagName(s)[0]; 
    if (d.getElementById(id)) 
      return;
    js = d.createElement(s); js.id = id; 
    js.src = "http://connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
    fjs.parentNode.insertBefore(js, fjs);
  }(document, 'script', 'facebook-jssdk'));
</script>
</div>
</div>
```

• **webhelp.footer.include** - Specifies whether or not to include footer in each WebHelp page. Possible values: 'yes', 'no'. If set to 'no' no footer is added to the WebHelp pages. If set to 'yes' and the webhelp.footer.file parameter has a value, then the content of that file is used as footer. If the webhelp.footer.file has no value then the default Oxygen footer is inserted in each WebHelp page.
• **args.default.language** - If the language is not detected in the DITA map, this parameter is used. The default sample value is `en-us`.

• **webhelp.search.japanese.dictionary** - The file path of the user dictionary that will be used by the Kuromoji morphological indexer that is used for indexing Japanese content in the WebHelp pages.

Once Oxygen XML Developer plugin finishes the transformation process, the output is automatically opened in your default browser.

**How to Localize the Interface of WebHelp Output**

Static labels that are used in the WebHelp output are kept in translation files in the

```bash
[OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization
```
folder. By default, the DITA-OT folder is ```[OXYGEN_DIR]/frameworks/dita/DITA-OT``` or possibly elsewhere if you are using a different DITA-OT distribution. Translation files have the `strings-lang1-lang2.xml` name format, where `lang1` and `lang2` are ISO language codes. For example, the US English text is kept in the ```strings-en-us.xml``` file.

Follow these steps to localize the interface of the WebHelp output:

1. Look for the `strings-[lang1]-[lang2].xml` file in

```bash
[OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization
```
(for example, the Canadian French file would be: `strings-fr-ca.xml`). If it does not exist, create one starting from `strings-en-us.xml`.

2. Translate all the labels from the above language file. Labels are stored in XML elements that have the following format:

```xml
<str name="Label name">Caption</str>
```

3. Make sure that the new XML file that you created in the previous two steps is listed in the file

```bash
[OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization/strings.xml
```
In our example for the Canadian French file, it should be listed as: ```<lang xml:lang="fr-ca" filename="strings-fr-ca.xml"/>
```

4. Edit the DITA Map WebHelp/DITA Map WebHelp with Feedback transformation scenario and set the ```args.default.language``` parameter to the code of the language you want to localize (for example, `fr-ca` for Canadian French).

5. Run the transformation scenario to produce the WebHelp output.

**Support for Right-to-Left (RTL) Oriented Languages**

To activate support for RTL languages, edit the DITA Map and set the `xml:lang` attribute on its root element (`map`). The corresponding attribute value can be set for following RTL languages:

- **ar-eg** - Arabic
- **he-il** - Hebrew
- **ur-pk** - Urdu

**WebHelp Search Engine Optimization**

A DITA WebHelp transformation scenario can be configured to produce a `sitemap.xml` file that is used by search engines to aid crawling and indexing mechanisms. A `sitemap` lists all pages of a WebHelp system and allows webmasters to provide additional information about each page, such as the date it was last updated, change frequency, and importance of each page in relation to other pages in your WebHelp deployment.

The structure of the `sitemap.xml` file looks like this:

```xml
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/topics/introduction.html</loc>
    <lastmod>2014-10-24</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
  <url>
    <loc>http://www.example.com/topics/care.html</loc>
    <lastmod>2014-10-24</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
  ...
</urlset>
```
Each page has a `<url>` element structure containing additional information, such as:

- **loc** - the URL of the page. This URL must begin with the protocol (such as http), if required by your web server. It is constructed from the value of the `webhelp.sitemap.base.url` parameter from the transformation scenario and the relative path to the page (collected from the `href` attribute of a `topicref` element in the DITA map).
  
  **Note:** The value must have less than 2,048 characters.

- **lastmod** - the date when the page was last modified. The date format is `YYYY-MM-DD`.

- **changefreq** - indicates how frequently the page is likely to change. This value provides general information to assist search engines, but may not correlate exactly to how often they crawl the page. Valid values are: always, hourly, daily, weekly, monthly, yearly, and never. The first time the `sitemap.xml` file is generated, the value is set based upon the value of the `webhelp.sitemap.change.frequency` parameter in the DITA WebHelp transformation scenario. You can change the value in each `url` element by editing the `sitemap.xml` file.
  
  **Note:** The value `always` should be used to describe documents that change each time they are accessed. The value `never` should be used to describe archived URLs.

- **priority** - the priority of this page relative to other pages on your site. Valid values range from 0.0 to 1.0. This value does not affect how your pages are compared to pages on other sites. It only lets the search engines know which pages you deem most important for the crawlers. The first time the `sitemap.xml` file is generated, the value is set based upon the value of the `webhelp.sitemap.priority` parameter in the DITA WebHelp transformation scenario. You can change the value in each `url` element by editing the `sitemap.xml` file.
  
  **Note:** lastmod, changefreq, and priority are optional elements.

### Creating and Editing the `sitemap.xml` File

Follow these steps to produce a `sitemap.xml` file for your WebHelp system, which can then be edited to fine-tune search engine optimization:

1. **Edit** the transformation scenario you currently use for obtaining your WebHelp output. This opens the **Edit DITA Scenario** dialog.
2. Open the **Parameters** tab and set a value for the following parameters:
   - `webhelp.sitemap.base.url` - the URL of the location where your WebHelp system is deployed
     
     **Note:** This parameter is required in order for Oxygen XML Developer plugin to generate the `sitemap.xml` file.
   - `webhelp.sitemap.change.frequency` - how frequently the WebHelp pages are likely to change (accepted values are: always, hourly, daily, weekly, monthly, yearly, and never)
   - `webhelp.sitemap.priority` - the priority of each page (value ranging from 0.0 to 1.0)
3. Execute the transformation scenario.
4. Look for the `sitemap.xml` file in the transformation's output folder. Edit the file to fine-tune the parameters of each page, according to your needs.

### Indexing Japanese Content in WebHelp Pages

To optimize the indexing of Japanese content in WebHelp pages, the Kuromoji analyzer can be used. This analyzer is not included in the Oxygen XML Developer plugin installation kit and must be downloaded and added.

To use the Kuromoji analyzer to index Japanese content in your WebHelp system, follow these steps:

1. Download the analyzer jar file from `http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0`
2. Place the Kuromoji analyzer jar file in the following directory: `[OXYGEN INSTALLATION DIRECTORY]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.webhelp/lib`
3. For the analyzer to work properly, search terms that are entered into your WebHelp pages must be separated by spaces.

Optionally a Japanese user dictionary can be set with the `webhelp.search.japanese.dictionary` parameter.

**Compiled HTML Help (CHM) Output Format**

To perform a Compiled HTML Help (CHM) transformation Oxygen XML Developer plugin needs Microsoft HTML Help Workshop to be installed on your computer. Oxygen XML Developer plugin automatically detects HTML Help Workshop and uses it.

**Note:** HTML Help Workshop might fail if the files used for transformation contain accents in their names, due to different encodings used when writing the `.hhp` and `.hhc` files. If the transformation fails to produce the CHM output but the `.hhp` (HTML Help Project) file is already generated, you can manually try to build the CHM output using HTML Help Workshop.

**Changing the Output Encoding**

Oxygen XML Developer plugin uses the `htmlhelp.locale` parameter to correctly display specific characters of different languages in the output of the Compiled HTML Help (CHM) transformation. The Compiled HTML Help (CHM) default scenario that comes bundled with Oxygen XML Developer plugin has the `htmlhelp.locale` parameter set to `en-US`.

The default value of the `htmlhelp.locale` is `en-US`. To customize this parameter, go to Configure Transformation Scenarios and click the Edit button. In the parameter tab search for the `htmlhelp.locale` parameter and change its value to the desired language tag.

The format of the `htmlhelp.locale` parameter is `LL-CC`, where `LL` represents the language code (en for example) and `CC` represents the country code (US for example). The language codes are contained in the ISO 639-1 standard and the country codes are contained in the ISO 3166-1 standard. For further details about language tags, go to http://www.rfc-editor.org/rfc/rfc5646.txt.

**Kindle Output Format**

Oxygen XML Developer plugin requires KindleGento generate Kindle output from DITA Maps. To install KindleGen for use by Oxygen XML Developer plugin, follow these steps:

1. Go to www.amazon.com/kindleformat/kindlegen and download the zip file that matches your operating system.
2. Unzip the file on your local disk.
3. Start Oxygen XML Developer plugin and open a DITA Map in the DITA Maps Manager view.
4. In the DITA Maps Manager View open the Configure Transformation Scenario(s) dialog box.
5. Select the DITA Map Kindle transformation and press the Edit button to edit it.
6. Go to Parameters tab and set the `kindlegen.executable` parameter as the path to the KindleGen directory.
7. Accept the changes.

**Migrating OOXML Documents to DITA**

Oxygen XML Developer plugin integrates the entire DITA for Publishers plugins suite, enabling you to migrate content from Open Office XML documents to DITA:

- Open an OOXML document in Oxygen XML Developer plugin. The document is opened in the Archive Browser view.
- From the Archive Browser, open `document.xml`.

**Note:** `document.xml` holds the content of the document.

- Click Configure Transformation Scenario(s) on the toolbar and apply the DOCX DITA scenario. If you encounter any issues with the transformation, click the link below for further details about the Word to DITA Transformation Framework.
DITA Map Templates
The default templates available for DITA maps are stored in \[OXYGEN_DIR\]/frameworks/dita/templates/map folder.

Here are some of the DITA Map templates available when creating new documents from templates:

- **DITA Map - Bookmap** - New DITA Bookmap.
- **DITA Map - Map** - New DITA Map.
- **DITA Map - Learning Map** - New DITA learning and training content specialization map.
- **DITA Map - Learning Bookmap** - New DITA learning and training content specialization bookmap.
- **DITA Map - Eclipse Map** - IBM specialization of DITA Map used for producing Eclipse Help plugins.

DITA for Publishers Map specialization templates:
- **D4P Map** - New DITA for Publishers Map.
- **D4P Pub-component-map** - New DITA for Publishers pub-component-map.
- **D4P Pubmap** - New DITA for Publishers pubmap.

The XHTML Document Type
The Extensible HyperText Markup Language (XHTML), is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is html.

The default schemas used for these documents are stored in the following locations:

- **XHTML 1.0** - \[OXYGEN_DIR\]/frameworks/xhtml/dtd/ or \[OXYGEN_DIR\]/frameworks/xhtml/nvdl/.
- **XHTML 1.1** - \[OXYGEN_DIR\]/frameworks/xhtml11/dtd/ or \[OXYGEN_DIR\]/frameworks/xhtml11/schema/.
- **XHTML 5** - \[OXYGEN_DIR\]/frameworks/xhtml/xhtml5 (epub3)/.

The default catalogs for the XHTML document type are as follows:

- \[OXYGEN_DIR\]/frameworks/xhtml/dtd/xhtmlcatalog.xml
- \[OXYGEN_DIR\]/frameworks/relaxng/catalog.xml
- \[OXYGEN_DIR\]/frameworks/nvdl/catalog.xml
- \[OXYGEN_DIR\]/frameworks/xhtml11/dtd/xhtmlcatalog.xml
- \[OXYGEN_DIR\]/frameworks/xhtml11/schema/xhtmlcatalog.xml
- \[OXYGEN_DIR\]/xhtml5 (epub3)/catalog-compat.xml

XHTML Transformation Scenarios
The following default transformation scenarios are available for XHTML:

- **XHTML to DITA concept** - Converts an XHTML document to a DITA concept document.
- **XHTML to DITA reference** - Converts an XHTML document to a DITA reference document.
- **XHTML to DITA task** - Converts an XHTML document to a DITA task document.
- **XHTML to DITA topic** - Converts an XHTML document to a DITA topic document.

XHTML Templates
Default templates are available for XHTML. They are stored in \[OXYGEN_DIR\]/frameworks/xhtml/templates folder and they can be used for easily creating basic XHTML documents.

Here are some of the XHTML templates available when creating new documents from templates:

- **XHTML - 1.0 Strict** - New Strict XHTML 1.0
- **XHTML - 1.0 Transitional** - New Transitional XHTML 1.0
- **XHTML - 1.1 DTD Based** - New DTD based XHTML 1.1
The TEI ODD Document Type

The Text Encoding Initiative - One Document Does it all (TEI ODD) is a TEI XML-conformant specification format that allows you to create a custom TEI P5 schema in a literate programming fashion. A system of XSLT stylesheets called Roma was created by the TEI Consortium for manipulating the ODD files.

A file is considered to be a TEI ODD document when the following conditions are true:

- The file extension is .odd.
- The document namespace is http://www.tei-c.org/ns/1.0.

The default schema, tei_odds.rng, used for these documents is stored in [OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/.

There are two default catalogs for the TEI ODD document type:

- [OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/catalog.xml
- [OXYGEN_DIR]/frameworks/tei/xml/tei/schema/catalog.xml

TEI ODD Transformation Scenarios

The following default transformations are available:

- **TEI ODD XHTML** - Transforms a TEI ODD document into an XHTML document
- **TEI ODD PDF** - Transforms a TEI ODD document into a PDF document using the Apache FOP engine
- **TEI ODD EPUB** - Transforms a TEI ODD document into an EPUB document
- **TEI ODD DOCX** - Transforms a TEI ODD document into a DOCX document
- **TEI ODD ODT** - Transforms a TEI ODD document into an ODT document
- **TEI ODD RelaxNG XML** - Transforms a TEI ODD document into a RelaxNG XML document
- **TEI ODD to DTD** - Transforms a TEI ODD document into a DTD document
- **TEI ODD to XML Schema** - Transforms a TEI ODD document into an XML Schema document
- **TEI ODD to RelaxNG Compact** - Transforms a TEI ODD document into a RelaxNG Compact document

TEI ODD Templates

There is only one default template which is stored in the [OXYGEN_DIR]/frameworks/tei/templates/TEI ODD folder and can be used for easily creating a basic TEI ODD document. This template is available when creating new documents from templates.

- **TEI ODD** - New TEI ODD document

The TEI P4 Document Type

The Text Encoding Initiative (TEI) Guidelines is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P4 document when one of the following conditions are true:

- The local name of the root is TEI.2.
- The public id of the document is -//TEI P4.

The default DTD schema, tei2.dtd, used for these documents is stored in [OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/.

The default catalogs for the TEI P4 document type are as follows:

- [OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/catalog.xml
- [OXYGEN_DIR]/frameworks/tei/xml/teip4/custom/schema/dtd/catalog.xml
TEI P4 Transformation Scenarios

The following default transformations are available:

- **TEI HTML** - Transforms a TEI document into an HTML document;
- **TEI P4 -> TEI P5 Conversion** - Convert a TEI P4 document into a TEI P5 document;
- **TEI PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine.

TEI P4 Templates

The default templates are stored in `[OXYGEN_DIR]/frameworks/tei/templates/TEI P4` folder and they can be used for easily creating basic TEI P4 documents. These templates are available when creating new documents from templates.

- **TEI P4 - Lite** - New TEI P4 Lite
- **TEI P4 - New Document** - New TEI P4 standard document

The TEI P5 Document Type

The Text Encoding Initiative (TEI) Guidelines is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P5 document when one of the following conditions are true:

- The document namespace is `http://www.tei-c.org/ns/1.0`.
- The public id of the document is `-//TEI P5`.

The default schemas used for these documents are stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/dtd/` or `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/`.

The default catalogs for the TEI P5 document type are as follows:

- `[OXYGEN_DIR]/frameworks/tei/xml/tei/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/tei/stylesheet/catalog.xml`

To watch our video demonstration about TEI editing, go to [http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html](http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html).

TEI P5 Transformation Scenarios

The following default transformations are available:

- **TEI P5 XHTML** - transforms a TEI P5 document into an XHTML document;
- **TEI P5 PDF** - transforms a TEI P5 document into a PDF document using the Apache FOP engine;
- **TEI EPUB** - transforms a TEI P5 document into an EPUB output. The EPUB output will contain any images referenced in the TEI XML document;
- **TEI DOCX** - transforms a TEI P5 document into a DOCX (OOXML) document. The DOCX document will contain any images referenced in the TEI XML document;
- **TEI ODT** - transforms a TEI P5 document into an ODT (ODF) document. The ODT document will contain any images referenced in the TEI XML document.

TEI P5 Templates

The default templates are stored in `[OXYGEN_DIR]/frameworks/tei/templates/TEI P5` folder and they can be used for easily creating basic TEI P5 documents. These templates are available when creating new documents from templates:

- **TEI P5 - All** - New TEI P5 All;
- **TEI P5 - Bare** - New TEI P5 Bare;
Customization of TEI Frameworks Using the Compiled Sources

The following procedure describes how to update to the latest stable version of TEI Schema and TEI XSL, already integrated in the TEI framework for Oxygen XML Developer plugin.

2. Go to Downloads;
3. Download the latest uploaded .zip file;
4. Unpack the .zip file and copy its content in the Oxygen XML Developer plugin frameworks folder.

The JATS Document Type

The JATS (NISO Journal Article Tag Suite) document type is a technical standard that defines an XML format for scientific literature.

A file is considered to be a JATS document when the PUBLIC ID of the document contains the string //NLM//DTD.

The default schemas for the JATS document types are stored in [OXYGEN_DIR]/frameworks/jats/O2-DTD/.

The default XML catalog, JATS-catalog-O2.xml, is stored in [OXYGEN_DIR]/frameworks/O2-DTD/.

JATS Transformation Scenarios

The following default transformation scenario is available for JATS documents:

• **JATS Preview (simple HTML)** - Converts a JATS document to a simple HTML document.

JATS Templates

Default templates are available for JATS documents. They are stored in [OXYGEN_DIR]/frameworks/jats/templates folder and they can be used for easily creating basic JATS documents.

The default JATS templates that are available when creating new documents from templates are as follows:

• **Archiving** - JATS archiving tag set version 1.0.
• **Authoring** - JATS authoring tag set version 1.0.
• **Book** - JATS book tag set version 1.0.
• **Publishing** - JATS publishing tag set version 1.0.

The EPUB Document Type

Three distinct frameworks are supported for the EPUB document type:

• **NCX** - A declarative global navigation definition.
• **OCF** - The Open Container Format (OCF) defines a mechanism by which all components of an Open Publication Structure (OPS) can be combined into a single file system entity.
• **OPF** - The Open Packaging Format (OPF) defines the mechanism by which all components of a published work that conforms to the Open Publication Structure (OPS) standard (including metadata, reading order, and navigational information) are packaged in an OPS Publication.

Note: Oxygen XML Developer plugin supports both OPF 2.0 and OPF 3.0.
### Document Templates

The default templates for the *NCX* and *OCF* document types are located in the 
`[OXYGEN_DIR]/frameworks/docbook/templates` folder.

The default template for the *OPF 2.0* document type is located in the 
`[OXYGEN_DIR]/frameworks/docbook/templates/2.0` folder.

The default template for the *OPF 3.0* document type is located in the 
`[OXYGEN_DIR]/frameworks/docbook/templates/3.0` folder.

The following EPUB templates are available when creating *new documents from templates*:

- **NCX - Toc** - New table of contents.
- **OCF - Container** - New container based OCF.
- **OCF - Encryption** - New encryption based OCF.
- **OCF - Signatures** - New signature based OCF.
- **OPF 2.0 - Content (2.0)** - New OPF 2.0 content.
- **OPF 3.0 - Content (3.0)** - New OPF 3.0 content.

### The DocBook Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with the DocBook olink.

A file is considered to be a *Targetset* when the root name is `targetset`.

The default schema, `targetdatabase.dtd`, for this type of document is stored in 
`[OXYGEN_DIR]/frameworks/docbook/xsl/common/`.

### Document Templates

The default template for DocBook *Targetset* documents is located in the 
`[OXYGEN_DIR]/frameworks/docbook/templates/Targetset` folder.

The following DocBook *Targetset* template is available when creating *new documents from templates*:

Chapter 7

Transforming Documents

Topics:
- Transformation Scenarios
- Output Formats

XML documents can be transformed into a variety of user-friendly output formats that can be viewed by other users. This process is known as a *transformation*.
Transformation Scenarios

A transformation scenario is a set of complex operations and settings that gives you the possibility to obtain outputs of multiple types (XML, HTML, PDF, EPUB, etc.) from the same source of XML files and stylesheets.

Executing a transformation scenario implies multiple actions, such as:

- Validating the input file.
- Obtaining intermediate output files (for example, formatting objects for the XML to PDF transformation).
- Using transformation engines to produce the output.

Before transforming an XML document in Oxygen XML Developer plugin, you need to define a transformation scenario to apply to that document. A scenario is a set of values for various parameters that define a transformation. It is not related to a particular document, but rather to a document type. Types of transformation scenarios include:

- **Scenarios that Apply to XML Files** - This type of scenario contains the location of an XSLT stylesheet that is applied on the edited XML document, as well as other transformation parameters.
- **Scenarios that Apply to XSLT Files** - This type of scenario contains the location of an XML document, on which the edited XSLT stylesheet is applied, as well as other transform parameters.
- **Scenarios that Apply to XQuery Files** - This type of scenario contains the location of an XML source, on which the edited XQuery file is applied, as well as other transform parameters. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()` . When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- **Scenarios that Apply to SQL Files** - This type of scenario specifies a database connection for the database server that runs the SQL file that is associated with the scenario. The data processed by the SQL script is located in the database.
- **Scenarios that Apply to XProc Files** - This type of scenario contains the location of an XProc script, as well as other transform parameters.
- **DITA-OT Scenarios** - This type of scenario provides the parameters for an ANT transformation that executes a DITA-OT build script. Oxygen XML Developer plugin comes with a built-in version of ANT and a built-in version of DITA-OT, although you can also set other versions in the scenario.
- **ANT Scenarios** - This type of scenario contains the location of an ANT build script, as well as other transform parameters.

**Note:**

Status messages generated during the transformation process are displayed in the **Console view**.

Defining a New Transformation Scenario

Defining a transformation scenario is the first step in the process of transforming a document. The following types of scenarios are available:

- **XML Transformation with XSLT** - Specifies the transformation parameters and location of an XSLT stylesheets that is applied to the edited XML document. This scenario is useful when you develop an XML document and the XSLT document is in its final form.
- **XML Transformation with XQuery** - Specifies the transform parameters and location of an XQuery file that is applied to the edited XML document.
- **DITA-OT Transformation** - Specifies the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Developer plugin comes with a built-in version of Ant and a built-in version of DITA-OT but different versions can be set in the scenario.
- **ANT Transformation** - Allows you to configure the options and parameters of an ANT build script.
- **XSLT Transformation** - Specifies the transformation parameters and location of an XML document to which the edited XSLT stylesheets is applied. This scenario is useful when you develop an XSLT document and the XML document is in its final form.
- **XProc Transformation** - Specified the transformation parameters and location of an XProc script.
• **XQuery Transformation** - Specifies the transformation parameters and location of an XML source to which the edited XQuery file is applied. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()` . When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.

• **SQL Transformation** - Specifies a database connection for the database server that runs the SQL file associated with the scenario. The data processed by the SQL script is located in the database.

**XML transformation with XSLT**

To create an **XML transformation with XSLT** scenario, use one of the following methods:

• Go to **Window > Show View** and select Transformation Scenarios to display this view. Click the **New** button and select XML transformation with XSLT.

• Use the **Configure Transformation Scenario(s)** ([Alt Shift T, C (Command Alt T, C on OS X)]) action from the Transformation toolbar or the XML menu. Then click the **New** button and select XML transformation with XSLT.

• Use the **Apply Transformation Scenario(s)** ([Alt Shift T, T (Command Alt T, T on OS X)]) action from the Transformation toolbar or the XML menu. Then click the **New** button and select XML transformation with XSLT.

  **Note:** If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

• **XSLT**.

• **FO Processors**.

• **Output**.

**The XSLT Tab**

The **XSLT** tab contains the following options:

• **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  **Note:** If the transformer engine is Saxon 9 and a custom URI resolver is configured in Preferences for Saxon 9, the XML input of the transformation is passed to that URI resolver.

  **Note:** If the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and the name of an initial template is specified in the scenario, the XML URL field can be empty. The XML URL field can also be empty if you use external XSLT processors. Otherwise, a value is mandatory in the XML URL field.

• **XSL URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

• **Insert Editor Variables**
  
  Opens a pop-up menu allowing you to introduce special **Oxygen XML Developer plugin editor variables** or **custom editor variables** in the XML URL field.

• **Browse for local file**
  
  Opens a local file browser dialog box allowing you to select a local file.
Browse workspace
Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file
Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file
Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer
Opens the Data Source Explorer window.

Search for file
Allows you to find a file in the current project.

The rest of the options available in the XSLT tab allow you to further customize the transformation scenario:

• **Use "xml-stylesheet" declaration** - Use the stylesheet declared with an xml-stylesheet declaration instead of the stylesheet specified in the XSL URL field. By default, this checkbox is not selected and the transformation applies the XSLT stylesheet that is specified in the XSL URL field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the xml-stylesheet processing instruction.

• **Transformer** - This drop-down list presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These are the built-in engines and the external engines defined in the Custom Engines preferences page. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).

  • **Advanced options** - Allows you to configure the advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the Saxon-HE/PE/EE preferences page. For the current transformation scenario, these Advanced options override the options configured in the Saxon-HE/PE/EE preferences page. The Initial mode and template option is only available in the Advanced options. It is a Saxon-specific option that sets the name of the first XSLT template that starts the XSLT transformation or the initial mode of the transformation.

• **Parameters** - Opens the Configure parameters dialog, allowing you to configure the XSLT parameters used in the current transformation. In this dialog you can also configure the parameters of additional stylesheets by using the Additional XSLT stylesheets button. If the XSLT transformation engine is custom-defined, you can not use this dialog to configure the parameters sent to the custom engine. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line that starts the transformation process.

• **Extensions** - Opens the dialog for configuring the XSLT/XQuery extension jars or classes that define extension Java functions or extension XSLT elements used in the transformation.

• **Additional XSLT stylesheets** - Opens the dialog for adding XSLT stylesheets that are applied on the main stylesheet that is specified in the XSL URL field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

Configure XSLT Parameters
The global parameters of the XSLT stylesheet used in a transformation scenario can be configured by using the Parameters button in the XSLT tab of a new or edited transformation scenario dialog.

The table displays all the parameters of the current XSLT stylesheet, all imported and included stylesheets, and all additional stylesheets, along with their descriptions and current values. You can also add, edit, and remove parameters. Use the Filter text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

If the XPath column is checked, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.
For example, you can use expressions such as:

```xml
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 (such as `${cfdu}` [current file directory]) to specify other locations:
   
   ```xml
doc('${cfdu}/test.xml')/*
   ```

2. You cannot use XSLT Functions. Only XPath functions are allowed.

The following actions are available for managing the parameters:

**New**

Opens the Add Parameter dialog that allows you to add a new parameter to the list. An editor variable can be inserted in the text box using the Insert Editor Variables button. If the Evaluate as XPath option is enabled, the parameter will be evaluated as an XPath expression.

**Edit**

Opens the Edit Parameter dialog that allows you to edit the selected parameter. An editor variable can be inserted in the text box using the Insert Editor Variables button. If the Evaluate as XPath option is enabled, the parameter will be evaluated as an XPath expression.

**Unset**

Resets the selected parameter to its default value. Available only for edited parameters with set values.

**Delete**

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The bottom panel presents the following:

• The default value of the parameter selected in the table.
• A description of the parameter, if available.
• The system ID of the stylesheet that declares it.

**XSLT/XQuery Extensions**

The Libraries dialog box is used to specify the jars and classes that contain extension functions called from the XSLT or XQuery file of the current transformation scenario.

An extension function called from the XSLT or XQuery file of the current transformation scenario will be searched, in the specified extensions, in the order of the list displayed in this dialog. To change the order of the items, select the item to be moved and press the Move up or Move down buttons.

**The FO Processor Tab**

The FO Processor tab contains the following options:

• Perform FO Processing - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in Preferences) during the transformation.
• XSLT result as input - The FO processor is applied to the result of the XSLT transformation that is defined in the XSLT tab.
• XML URL as input - The FO processor is applied to the input XML file.
• Method - The output format of the FO processing. Available options depend on the selected processor type.
• Processor - Specifies the FO processor. It can be the built-in Apache FOP processor or an external processor.

**The Output Tab**

The Output tab contains the following options:
• **Prompt for file** - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.

• **Save As** - The path of the file where the result of the transformation is stored. The path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the 📃 Insert Editor Variables button.

• **Open in Browser/System Application** - If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the Acrobat Reader application).

  **Note:** To set the web browser that is used for displaying HTML/XHTML pages, open the Preferences dialog box, then go to General > Web Browser.

  • **Saved file** - When Open in Browser/System Application is selected, this button can be used to specify that Oxygen XML Developer plugin automatically opens the file specified in the Save As text field at the end of the transformation.

  • **Other location** - When Open in System Application is selected, this option can be used to specify that Oxygen XML Developer plugin opens the file specified here. The file path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the 📃 Insert Editor Variables button.

• **Open in editor** - When this is enabled, the transformation result specified in the Save As field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).

• **Show in results view as**

  • **XHTML** - Can only be enabled if Open in Browser/System Application is disabled. If this is checked, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the Open in browser option.

  • **XML** - If this is checked, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting, specific for XML documents.

• **Image URLs are relative to** - If Show in results view as XHTML is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

**Additional XSLT Stylesheets**

The list of additional XSLT stylesheets can be edited in the dialog box opened by the Additional XSLT Stylesheets button in the XSLT tab of a new or edited transformation scenario dialog box. The following actions are available:

**Add**

Adds a stylesheet in the Additional XSLT stylesheets list using a file browser dialog box. You can type an editor variable in the file name field of the browser dialog box. The name of the stylesheet will be added in the list after the current selection.

**Remove**

Deletes the selected stylesheet from the Additional XSLT stylesheets list.

**Up**

Moves the selected stylesheet up in the list.

**Down**

Moves the selected stylesheet down in the list.
XML Transformation with XQuery

Use the XML transformation with XQuery scenario to apply a transformation in which an XQuery file queries an XML file for the output results.

To create an XML transformation with XQuery scenario, use one of the following methods:

- Go to Window > Show View and select Transformation Scenarios to display this view. Click the New button and select XML transformation with XQuery.
- Use the Configure Transformation Scenario(s) (Alt Shift T, C (Command Alt T, C on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select XML transformation with XQuery.
- Use the Apply Transformation Scenario(s) (Alt Shift T, T (Command Alt T, T on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select XML transformation with XQuery.

Note: If a scenario is already associated with the edited document, selecting Apply Transformation Scenario(s) runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the Apply Transformation Scenario button.

All three methods open the New Scenario dialog box. This dialog allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

- XQuery
- FO Processor
- Output

The XQuery Tab

The XQuery tab contains the following options:

- XML URL - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in Preferences for Saxon 9, the XML input of the transformation is passed to that URI resolver.

- XQuery URL - specifies the source XQuery file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the XML URL and XQuery URL fields:

- Insert Editor Variables
  Opens a pop-up menu allowing you to introduce special Oxygen XML Developer plugin editor variables or custom editor variables in the XML URL field.

- Browse for local file
  Opens a local file browser dialog box allowing you to select a local file.

- Browse workspace
  Opens a file browser dialog box allowing you to select a file from the local workspace.

- Browse for remote file
  Opens an URL browser dialog box allowing you to select a remote file.

- Browse for archived file
  Opens a zip archive browser dialog box allowing you to select a file from a zip archive.
Browse Data Source Explorer
Opens the Data Source Explorer window.

Search for file
Allows you to find a file in the current project.

The rest of the options available in the XQuery tab allow you to further customize the transformation scenario:

• Transformer - This drop-down list presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These are the built-in engines and the external engines defined in the Custom Engines preferences page. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).

• Advanced options - configure advanced options specific for the Saxon HE/PE/EE engine.

• Parameters - Opens the Configure parameters dialog for configuring the XQuery parameters. You can use buttons in this dialog you can add, edit, or remove parameters. If the XQuery transformation engine is custom-defined you cannot use this dialog to set parameters. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XQuery engine to include the necessary parameters in the command line that starts the transformation process.

  Note: Use the Filter text box to search for a specific term in the entire parameters collection.

• Extensions - Opens the dialog for configuring the XSLT/XQuery extension jars or classes that define extension Java functions or extension XSLT elements used in the transformation.

XSLT/XQuery Extensions
The Libraries dialog box is used to specify the jars and classes that contain extension functions called from the XSLT or XQuery file of the current transformation scenario.

An extension function called from the XSLT or XQuery file of the current transformation scenario will be searched, in the specified extensions, in the order of the list displayed in this dialog. To change the order of the items, select the item to be moved and press the Move up or Move down buttons.

The FO Processor Tab
The FO Processor tab contains the following options:

• Perform FO Processing - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in Preferences) during the transformation.

• XQuery result as input - the FO processor is applied to the result of the XQuery transformation defined in the XQuery tab.

• XML URL as input - The FO processor is applied to the input XML file.

• Method - The output format of the FO processing. Available options depend on the selected processor type.

• Processor - Specifies the FO processor. It can be the built-in Apache FOP processor or an external processor.

The Output Tab
The Output tab contains the following options:

• Present as a sequence - Enabling this option will reduce the time necessary to fetch the full result, as it will only fetch the first chunk of the result.

• Prompt for file - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.

• Save As - The path of the file where the result of the transformation is stored. The path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the Insert Editor Variables button.
• **Open in Browser/System Application** - If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the Acrobat Reader application).

  **Note:** To set the web browser that is used for displaying HTML/XHTML pages, open the Preferences dialog box, then go to General > Web Browser.

• **Saved file** - When Open in Browser/System Application is selected, this button can be used to specify that Oxygen XML Developer plugin automatically opens the file specified in the Save As text field at the end of the transformation.

• **Other location** - When Open in System Application is selected, this option can be used to specify that Oxygen XML Developer plugin opens the file specified here. The file path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the Insert Editor Variables button.

• **Open in editor** - When this is enabled, the transformation result specified in the Save As field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).

• **Show in results view as**
  
  • **XHTML** - Can only be enabled if Open in Browser/System Application is disabled. If this is checked, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the Open in browser option.

  • **XML** - If this is checked, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting, specific for XML documents.

• **Image URLs are relative to** - If Show in results view as XHTML is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

### DITA OT Transformation

To create a DITA OT Transformation scenario, use one of the following methods:

• Go to Window > Show View and select Transformation Scenarios to display this view. Click the New button and select DITA OT Transformation.

• Use the Configure Transformation Scenario(s) (Alt Shift T, C (Command Alt T, C on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select DITA OT Transformation.

• Use the Apply Transformation Scenario(s) (Alt Shift T, T (Command Alt T, T on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select DITA OT Transformation.

  **Note:** If a scenario is already associated with the edited document, selecting Apply Transformation Scenario(s) runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the Apply Transformation Scenario button.

All three methods open the DITA Transformation Type dialog box that presents the list of possible outputs.
Figure 164: DITA Transformation Type Dialog Box

Select the desired type of output and click **OK**. This opens the **New Scenario** dialog box, which allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs (only those that are appropriate for the chosen output type will be displayed):

- **Skins** (Available for **WebHelp** and **WebHelp with Feedback** output types).
- **FO Processor** (Available for PDF output types).
- **Parameters**
- **Filters**
- **Advanced**
- **Output**

**The Skins Tab**

A *skin* is a collection of CSS properties that can alter the look of the output by changing colors, font types, borders, margins, and paddings. This allows you to rapidly adapt the look and feel of the output for your organization.

Oxygen XML Developer plugin provides a set of predefined *skins* for the **DITA Map WebHelp** and **DITA Map WebHelp with Feedback** transformation scenarios.
The predefined skins cover a wide range of chromatic themes, ranging from a very light one to a high-contrast variant. By default, the Oxygen skin is selected (notice the light blue border around the skin preview). If you want to obtain an output without any customization, deselect the currently selected skin.

To see how the skin looks when applied on a sample documentation project that is stored on the Oxygen XML Developer plugin website, press the Online preview link.

![Figure 165: The Skins Tab](image)

The output can also be styled by setting the args.css parameter in the Parameters tab. The properties taken from the stylesheet referenced in this parameter take precedence over the properties declared in the skin set in the Skins tab.

The FO Processor Tab

This tab allows you to select an FO Processor, when you choose to generate PDF output.
You can choose the following processors:

- **Apache FOP** - The default processor that comes bundled with.
- **XEP** - The RenderX XEP processor.

  If XEP is already installed, displays the detected installation path under the drop-down list. XEP is considered installed if it was detected in one of the following sources:
  
  - XEP was configured as an external FO Processor in the **FO Processors option page**.
  - The system property `com.oxygenxml.xep.location` was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows).
  - XEP was installed in the `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/lib` directory of the installation directory.

- **Antenna House** - The Antenna House AH (v5) or XSL (v4) Formatter processor.

  If Antenna House is already installed, displays the detected installation path under the drop-down list. Antenna House is considered installed if it was detected in one of the following sources:
  
  - Environment variable set by Antenna House installation (the newest installation version will be used, v5 being preferred over v4).
  - Antenna House was added as an external FO Processor in the preferences pages.

To further customize the PDF output obtained from the Antenna House processor:

- **Edit** the transformation scenario.
- Open the **Parameters tab**.
- Add the `env.AXF_OPT` parameter and point to Antenna House configuration file.
The Parameters Tab

The Parameters tab allows you to configure the parameters sent to the DITA-OT build file.

The table displays all the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: XHTML or PDF), along with their description and current values. You can find more information about each parameter in the DITA OT Documentation. You can also add, edit, and remove parameters. Use the Filter text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an editor variable selector to simplify setting a file path as the value of a parameter.

Note: To input parameter values at runtime, use the ask editor variable in the Value column.

The following actions are available for managing parameters:

New
Opens the Add Parameter dialog box that allows you to add a new parameter to the list. An editor variable can be inserted in the text box using the ✉️ Insert Editor Variables button.

Edit
Opens the Edit Parameter dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Unset
Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete
Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Filters Tab

The Filters tab allows you to add filters to remove certain content elements from the generated output.
There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the DITA map, you can specify the file to be used when filtering content. An *editor variable* can be inserted for the file path by using the *Insert Editor Variables* button. You can find out more about constructing a DITAVAL file in the [DITA OT Documentation](https://dita.ot).  

- **Exclude from output all elements with any of the following attributes** - By using the + New, ✎ Edit, or ✕ Delete buttons at the bottom of the pane, you can configure a list of attributes (name and value) to exclude all elements that contain any of these attributes from the output.

### The Advanced Tab

The **Advanced** tab allows you to specify advanced options for the transformation scenario.
You can specify the following parameters:

- **Custom build file** - If you use a custom DITA-OT build file, you can specify the path to the customized build file. If empty, the build.xml file from the dita.dir parameter that is configured in the Parameters tab is used. An editor variable can be inserted for the file path by using the `Insert Editor Variables` button.

- **Build target** - Optionally, you can specify a build target for the build file. If no target is specified, the default init target is used.

- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation (such as `-verbose`).

- **Ant Home** - You can choose between the default or custom ANT installation to run the transformation.

- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by .

- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by ANT. For example, if it is set to `-Xmx384m`, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (`OutOfMemoryError`).

- **Libraries** - By default, adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the ANT transformer.

**The Output Tab**

The **Output** tab allows you to configure options that are related to the location where the output is generated.
You can specify the following parameters:

- **Base directory** - All the relative paths that appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located. An editor variable can be inserted for the path by using the Insert Editor Variables button.

- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained. An editor variable can be inserted for the path by using the Insert Editor Variables button.

- **Output folder** - The folder where the content of the final output is stored. An editor variable can be inserted for the path by using the Insert Editor Variables button.

  **Note:** If the DITA map or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

- **Open in Browser/System Application** - If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the Acrobat Reader application).

  **Note:** To set the web browser that is used for displaying HTML/XHTML pages, open the Preferences dialog box, then go to General > Web Browser.

  - **Saved file** - When Open in Browser/System Application is selected, this button can be used to specify that Oxygen XML Developer plugin automatically opens the file specified in the Save As text field at the end of the transformation.

  - **Other location** - When Open in System Application is selected, this option can be used to specify that Oxygen XML Developer plugin opens the file specified here. The file path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the Insert Editor Variables button.
**Troubleshooting DITA Transformation Errors**

If a DITA transformation results in errors or warnings, the information is displayed in the message panel at the bottom of the editor. The information includes the severity, description of the problem, the name of the resource, and the path of the resource.

To help prevent and solve DITA transformation problems, follow these steps:

1. Validate your DITA documents by using the **Validate** action from the **Validation** toolbar drop-down list, the XML menu, or from the **Validate** menu when invoking the contextual menu in the Navigator view.
2. If this action results in validation errors, solve them prior to executing the transformation. Also, you should pay attention to the warning messages because they may identify problems in the transformation.
3. **Execute the DITA transformation scenario.**
4. If the transformation results in errors or warnings, they are displayed in the **Transformation problems** message panel at the bottom of the editor. The following information is presented to help you troubleshoot the problems:
   - **Severity** - The first column displays the following icons that indicate the severity of the problem:
     - **Informational** - The transformation encountered a condition of which you should be aware.
     - **Warning** - The transformation encountered a problem that should be corrected.
     - **Error** - The transformation encountered a more severe problem, and the output is affected or cannot be generated.
   - **Info** - You can click on the **See More** icon to open a web page that contains details about DITA-OT error messages.
   - **Description** - A description of the problem.
   - **Resource** - The name of the transformation resource.
   - **System ID** - The path of the transformation resource.
5. Use this information or other resources from the online DITA-OT community to solve the transformation problems before re-executing the transformation scenario.

**ANT Transformation**

An ANT transformation scenario is usually associated with an Ant build script. Oxygen XML Developer plugin runs an ANT transformation scenario as an external process that executes the Ant build script with the built-in Ant distribution (Apache Ant version 1.8.2) that comes with the application, or optionally with a custom Ant distribution configured in the scenario.

To create an **ANT transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **ANT transformation**.
- Use the **Configure Transformation Scenario(s)** action from the Transformation toolbar or the XML menu. Then click the **New** button and select **ANT transformation**.
- Use the **Apply Transformation Scenario(s)** action from the Transformation toolbar or the XML menu. Then click the **New** button and select **ANT transformation**.

**Note:** If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The dialog box contains the following tabs:
The Options Tab

The Options tab allows you to specify the following options:

- **Working directory** - The path of the current directory of the Ant external process. An *editor variable* can be inserted for the file path by using the ✉️ Insert Editor Variables button.
- **Build file** - The Ant script file that is the input of the Ant external process. An *editor variable* can be inserted for the file path by using the ✉️ Insert Editor Variables button.
- **Build target** - Optionally, you can specify a build target for the Ant script file. If no target is specified, the Ant target that is specified as the default in the Ant script file is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation (such as -verbose).
- **Ant Home** - You can choose between the default or custom ANT installation to run the transformation.
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Developer plugin.
- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by ANT. For example, if it is set to -Xmx384m, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (*OutOfMemoryError*).
- **Libraries** - By default, Oxygen XML Developer plugin adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the ANT transformer.

The Parameters Tab

The Parameters tab allows you to configure the parameters that are accessible as Ant properties in the Ant build script.

The table displays all the parameters that are available in the Ant build script, along with their description and current values. You can also add, edit, and remove parameters. Use the Filter text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.

⚠️ **Note:** To input parameter values at runtime, use the *ask editor variable* in the Value column.

The following actions are available for managing parameters:

**New**

Opens the Add Parameter dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the ✉️ Insert Editor Variables button.

**Edit**

Opens the Edit Parameter dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

**Delete**

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Output Tab

The Output tab contains the following options:
• **Open** - Allows you to specify the file to open automatically when the transformation is finished. Usually, this is the output file of the Ant process. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.

• **In System Application** - The file specified in the *Open* text box is opened in the system application that is set in the operating system as the default application for that type of file (for example, .pdf files are usually opened in the Acrobat Reader application).

• **In Editor** - The file specified in the *Open* text box is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor).

• The **Show console output** option allows you to specify when to display the console output log. The following options are available:
  - **When build fails** - displays the console output log if the build fails.
  - **Always** - displays the console output log, regardless of whether or not the build fails.

### XSLT Transformation

To create an **XSLT transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XSLT transformation**.

- Use the **Configure Transformation Scenario(s)** (Alt Shift T, C (Command Alt T, C on OS X)) action from the Transformation toolbar or the XML menu. Then click the **New** button and select **XSLT transformation**.

- Use the **Apply Transformation Scenario(s)** (Alt Shift T, T (Command Alt T, T on OS X)) action from the Transformation toolbar or the XML menu. Then click the **New** button and select **XSLT transformation**.

  **Note:** If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

- **XSLT**
- **FO Processors**
- **Output**

### The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  **Note:** If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

  **Note:** If the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and the name of an initial template is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty if you use *external XSLT processors*. Otherwise, a value is mandatory in the **XML URL** field.

- **XSL URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:
Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Developer plugin editor variables* or custom editor variables in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse workspace

Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

The rest of the options available in the XSLT tab allow you to further customize the transformation scenario:

- **Use "xml-stylesheet" declaration** - Use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the `XSL URL` field. By default, this checkbox is not selected and the transformation applies the XSLT stylesheet that is specified in the `XSL URL` field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction.

- **Transformer** - This drop-down list presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These are the built-in engines and the external engines defined in the Custom Engines preferences page. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).

- **Advanced options** - Allows you to configure the advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the Saxon-HE/PE/EE preferences page. For the current transformation scenario, these Advanced options override the options configured in the Saxon-HE/PE/EE preferences page. The Initial mode and template option is only available in the Advanced options. It is a Saxon-specific option that sets the name of the first XSLT template that starts the XSLT transformation or the initial mode of the transformation.

- **Parameters** - Opens the Configure parameters dialog, allowing you to configure the XSLT parameters used in the current transformation. In this dialog you can also configure the parameters of additional stylesheets by using the Additional XSLT stylesheets button. If the XSLT transformation engine is custom-defined, you can not use this dialog to configure the parameters sent to the custom engine. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line that starts the transformation process.

- **Extensions** - Opens the dialog for configuring the XSLT/XQuery extension jars or classes that define extension Java functions or extension XSLT elements used in the transformation.

- **Additional XSLT stylesheets** - Opens the dialog for adding XSLT stylesheets that are applied on the main stylesheet that is specified in the `XSL URL` field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

The FO Processor Tab

The FO Processor tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in Preferences) during the transformation.
• **XSLT result as input** - The FO processor is applied to the result of the XSLT transformation that is defined in the XSLT tab.

• **XML URL as input** - The FO processor is applied to the input XML file.

• **Method** - The output format of the FO processing. Available options depend on the selected processor type.

• **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

### The Output Tab

The **Output** tab contains the following options:

• **Prompt for file** - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.

• **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Developer plugin editor variables* or *custom editor variables* by using the 👉 **Insert Editor Variables** button.

• **Open in Browser/System Application** - If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the Acrobat Reader application).

⚠️ **Note:** To set the web browser that is used for displaying HTML/XHTML pages, *open the Preferences dialog box*, then go to **General > Web Browser**.

  • **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Developer plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.

  • **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Developer plugin opens the file specified here. The file path can include *special Oxygen XML Developer plugin editor variables* or *custom editor variables* by using the 👉 **Insert Editor Variables** button.

• **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).

• **Show in results view as**

  • **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    😡 **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

  • **XML** - If this is checked, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.

• **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

### XProc Transformation

A sequence of transformations described by an XProc script can be executed with an XProc transformation scenario. To create an **XProc transformation** scenario, use one of the following methods:

• Go to **Window > Show View** and select 👉 **Transformation Scenarios** to display this view. Click the **New** button and select **XProc transformation**.

• Use the 👉 **Configure Transformation Scenario(s)** (Alt Shift T, C (Command Alt T, C on OS X)) action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XProc transformation**.
Use the **Apply Transformation Scenario(s)** (Alt Shift T, T (Command Alt T, T on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select XProc transformation.

Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the New Scenario dialog box. This dialog allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs:

- **XProc**
- **Inputs**
- **Parameters**
- **Outputs**
- **Options**

### The XProc Tab

The XProc tab contains the following options:

- **XProc URL**: Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter value in the XProc URL:

- **Insert Editor Variables**: Opens a pop-up menu allowing you to introduce special Oxygen XML Developer plugin editor variables or custom editor variables in the XML URL field.

- **Browse for local file**: Opens a local file browser dialog box allowing you to select a local file.

- **Browse workspace**: Opens a file browser dialog box allowing you to select a file from the local workspace.

- **Browse for remote file**: Opens an URL browser dialog box allowing you to select a remote file.

- **Browse for archived file**: Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

- **Browse Data Source Explorer**: Opens the Data Source Explorer window.

- **Search for file**: Allows you to find a file in the current project.

- **Processor**: Allows you to select the XProc engine. You can select the built-in Calabash engine or a custom engine that is configured in the Preferences dialog.

### The Inputs Tab

The Inputs tab contains a list with the ports that the XProc script uses to read input data. Use the Filter text box to search for a specific term in the entire ports collection.

Each input port has an assigned name in the XProc script. The XProc engine reads data from the URL specified in the URL column. The **built-in editor variables** and **custom editor variables** can be used to specify the URL.

The following actions are available for managing the input ports:
New
Opens an **Edit** dialog that allows you to add a new port and its URL.

Edit
Opens an **Edit** dialog that allows you to modify the selected port and its URL.

Delete
Removes the selected port from the list. It is enabled only for new ports that have been added to the list.

**The Parameters Tab**
The **Parameters** tab presents a list of ports and parameters collected from the XProc script. The tab is divided into three sections:

- **List of Ports** - In this section you can use the **New** and **Delete** buttons to add or remove ports.
- **List of Parameters** - This section presents a list of parameters for each port and includes columns for the parameter name, namespace URI, and its value. Use the **Filter** text box to search for a specific term in the entire parameters collection. You can use the **New** and **Delete** buttons to add or remove parameters. You can edit the value of each cell in this table by double-clicking on the cell. You can also sort the parameters by clicking on the column headers.
- **Editor Variable Information** - The built-in editor variables and custom editor variables can be used for specifying the URI. The message pane at the bottom of the dialog provides more information about the editor variables that can be used.

**The Outputs Tab**
The **Outputs** tab displays a list of output ports (along with the URL) collected from the XProc script. Use the **Filter** text box to search for a specific term in the entire ports collection. You can also sort the columns by clicking on the column headers.

The following actions are available for managing the output ports:

New
Opens an **Edit** dialog that allows you to add a new output port and its URL. An editor variable can be inserted for the URL by using the **Insert Editor Variables** button. There is also a **Show in transformation results view** option that allows you to select whether or not the results will be displayed in the output results view.

Edit
Opens an **Edit** dialog that allows you to edit an existing output port and its URL. An editor variable can be inserted for the URL by using the **Insert Editor Variables** button. There is also a **Show in transformation results view** option that allows you to select whether or not the results will be displayed in the output results view.

Delete
Removes the selected output port from the list. It is enabled only for new ports that have been added to the list.

Additional options that are available at the bottom of this tab include:

Open in Editor
If this option is selected, the XProc transformation result is automatically opened in an editor panel.

Open in Browser/System Application
If this option is selected, you can specify a file to be opened at the end of the XProc transformation in the browser or system application that is associated with the file type. An editor variable can be inserted for the path by using the **Insert Editor Variables** button.

Results
The result of the XProc transformation can be displayed as a sequence in an output view with two sections:

- A list with the output ports on the left side.
- The content that correspond to the selected output port on the right side.
The Options Tab

The Options tab displays a list of the options collected from the XProc script. The tab is divided into two sections:

- **List of Options** - This section presents a list of options and includes columns for the option name, namespace URI, and its value. Use the Filter text box to search for a specific term in the entire options collection. You can use the **New** and **Delete** buttons to add or remove options. You can edit the value of each cell in this table by double-clicking on the cell. You can also sort the parameters by clicking on the column headers. The names of edited options are displayed in bold.

- **Editor Variable Information** - The built-in editor variables and custom editor variables can be used for specifying the URI. This section provides more information about the editor variables that can be used.

Configuring Calabash with XEP

To generate PDF output from your XProc pipeline (when using the Calabash XProc processor), follow these steps:

1. Open the \[OXYGEN_DIR\]/lib/xproc/calabash/engine.xml file.
2. Uncomment the `<system-property name="com.xmlcalabash.fo-processor" value="com.xmlcalabash.util.FoXEP"/` system property.
3. Uncomment the `<system-property name="com.renderx.xep.CONFIG" file="../../../../tools/xep/xep.xml"/>` system property. Edit the file attribute to point to the configuration file that is usually located in the XEP installation folder.
4. Uncomment the references to the XEP libraries. Edit them to point to the matching library names from the XEP installation directory.
5. Restart Oxygen XML Developer plugin.

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download from the application website as a zip file `xprocAPI.zip`. To create an XProc integration project, follow these steps:

1. Move the oxygen.jar file from \[OXYGEN_DIR\]/lib to the lib folder of your project.
2. Implement the `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface` interface.
3. Create a Java archive (jar) from the classes you created.
4. Create an `engine.xml` file according with the `engine.dtd` file. The attributes of the `engine` element are as follows:

   1. **name** - The name of the XProc engine.
   3. **class** - The complete name of the class that implements `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface`.
   4. **version** - The version of the integration.
   5. **engineVersion** - The version of the integrated engine.
   6. **vendor** - The name of the vendor / implementer.
7. `supportsValidation` - true if the engine supports validation, false otherwise.

The `engine` element has only one child, `runtime`. The `runtime` element contains several `library` elements with the `name` attribute containing the relative or absolute location of the libraries necessary to run this integration.

5. Create a folder with the name of the integration in the `[OXYGEN_DIR]/lib/xproc`.
6. Place the `engine.xml` and all the libraries necessary to run the new integration in that folder.

**XQuery Transformation**

To create an **XQuery transformation** scenario, use one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XQuery transformation**.
- Use the **Configure Transformation Scenario(s)** (`Alt Shift T, C (Command Alt T, C on OS X)`) action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XQuery transformation**.
- Use the **Apply Transformation Scenario(s)** (`Alt Shift T, T (Command Alt T, T on OS X)`) action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XQuery transformation**.

**Note:** If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs:

- **XQuery**
- **FO Processor**
- **Output**

**The XQuery Tab**

The **XQuery** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  **Note:** If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

- **XQuery URL** - specifies the source XQuery file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

- **Insert Editor Variables**
  
  Opens a pop-up menu allowing you to introduce special *Oxygen XML Developer plugin editor variables* or *custom editor variables* in the XML URL field.

- **Browse for local file**
  
  Opens a local file browser dialog box allowing you to select a local file.

- **Browse workspace**
  
  Opens a file browser dialog box allowing you to select a file from the local workspace.

- **Browse for remote file**
  
  Opens an URL browser dialog box allowing you to select a remote file.
Browse for archived file
Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer
Opens the Data Source Explorer window.

Search for file
Allows you to find a file in the current project.

The rest of the options available in the XQuery tab allow you to further customize the transformation scenario:

• Transformer - This drop-down list presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These are the built-in engines and the external engines defined in the Custom Engines preferences page. The engine you choose in this dialog is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support);

• Advanced options - configure advanced options specific for the Saxon HE / PE / EE engine.

• Parameters - Opens the Configure parameters dialog for configuring the XQuery parameters. You can use buttons in this dialog you can add, edit, or remove parameters. If the XQuery transformation engine is custom-defined you can not use this dialog to set parameters. Instead, you can copy all parameters from the dialog using contextual menu actions and edit the custom XQuery engine to include the necessary parameters in the command line that starts the transformation process.

Note: Use the Filter text box to search for a specific term in the entire parameters collection.

• Extensions - Opens the dialog for configuring the XSLT/XQuery extension jars or classes that define extension Java functions or extension XSLT elements used in the transformation.

The FO Processor Tab
The FO Processor tab contains the following options:

• Perform FO Processing - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in Preferences) during the transformation.

• XQuery result as input - the FO processor is applied to the result of the XQuery transformation defined in the XQuery tab.

• XML URL as input - The FO processor is applied to the input XML file.

• Method - The output format of the FO processing. Available options depend on the selected processor type.

• Processor - Specifies the FO processor. It can be the built-in Apache FOP processor or an external processor.

The Output Tab
The Output tab contains the following options:

• Present as a sequence - Enabling this option will reduce the time necessary to fetch the full result, as it will only fetch the first chunk of the result.

• Prompt for file - At the end of the transformation, a file browser dialog is displayed for specifying the path and name of the file that stores the transformation result.

• Save As - The path of the file where the result of the transformation is stored. The path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the Insert Editor Variables button.

• Open in Browser/System Application - If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, .pdf files are usually opened in the Acrobat Reader application).
Note: To set the web browser that is used for displaying HTML/XHTML pages, open the Preferences dialog box, then go to General > Web Browser.

- **Saved file** - When Open in Browser/System Application is selected, this button can be used to specify that Oxygen XML Developer plugin automatically opens the file specified in the Save As text field at the end of the transformation.
- **Other location** - When Open in System Application is selected, this option can be used to specify that Oxygen XML Developer plugin opens the file specified here. The file path can include special Oxygen XML Developer plugin editor variables or custom editor variables by using the Insert Editor Variables button.

- **Open in editor** - When this is enabled, the transformation result specified in the Save As field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
  - XHTML - Can only be enabled if Open in Browser/System Application is disabled. If this is checked, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.
    
  Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the Open in browser option.

- **XML** - If this is checked, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting, specific for XML documents.

- **Image URLs are relative to** - If Show in results view as XHTML is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

### SQL Transformation
To create an SQL transformation scenario, use one of the following methods:

- Go to Window > Show View and select Transformation Scenarios to display this view. Click the New button and select SQL transformation.
- Use the Configure Transformation Scenario(s) (Alt Shift T, C (Command Alt T, C on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select SQL transformation.
- Use the Apply Transformation Scenario(s) (Alt Shift T, T (Command Alt T, T on OS X)) action from the Transformation toolbar or the XML menu. Then click the New button and select SQL transformation.

Note: If a scenario is already associated with the edited document, selecting Apply Transformation Scenario(s) runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the Apply Transformation Scenario button.

All three methods open the New Scenario dialog box. This dialog allows you to configure the following options that control the transformation:

- **Name** - The unique name of the SQL transformation scenario.
- **SQL URL** - Allows you to specify the URL of the SQL script. You can use the following browsing buttons to enter value in this field:
Insert Editor Variables
Opens a pop-up menu allowing you to introduce special *Oxygen XML Developer plugin editor variables* or custom editor variables in the XML URL field.

Browse for local file
Opens a local file browser dialog box allowing you to select a local file.

Browse workspace
Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file
Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file
Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer
Opens the *Data Source Explorer* window.

Search for file
Allows you to find a file in the current project.

- **Connection** - Allows you to select a connection from a drop-down list. To configure a connection, use the **Advanced options** button to open the data source preferences page.
- **Parameters** - Allows you to configure the parameters of the transformation.

Configure Transformation Scenario(s) Dialog Box
You can use the **Configure Transformation Scenario(s)** dialog box to manage both the *built-in transformation scenarios* and the ones you create.

To open this dialog box, use the **Configure Transformation Scenario(s)** action from the Transformation toolbar or the XML menu.

![Configure Transformation Scenario(s) Dialog Box](image)

Figure 171: Configure Transformation Scenario(s) Dialog Box
The top section of the dialog box contains a filter that allows you to search through the scenarios list. The Settings button allows you to configure the following options:

- **Show all scenarios** - Select this option to display all the available scenarios, regardless of the document they are associated with.
- **Show only the scenarios available for the editor** - Select this option to only display the scenarios that Oxygen XML Developer plugin can execute for the current document type.
- **Show associated scenarios** - Select this option to only display the scenarios associated with the document you are editing.
- **Import scenarios** - This option opens the Import scenarios dialog box that allows you to select the scenarios file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:
  - Keep or replace the existing scenario.
  - Keep both scenarios.

  **Note:** When you keep both scenarios, Oxygen XML Developer plugin adds imported to the name of the imported scenario.

- **Export selected scenarios** - Use this option to export transformation and validation scenarios individually. Oxygen XML Developer plugin creates a scenarios file that contains the scenarios that you export.

The middle section of the dialog box displays the scenarios that you can apply to the current document. You can view both the scenarios associated with the current document type and the scenarios defined at project level. The following columns are used to display the transformation scenarios:

- **Association** - The check-boxes in this column mark whether a transformation scenario is associated with the current document.
- **Scenario** - This column presents the names of the transformation scenarios.
- **Type** - Displays the type of the transformation scenario. For further details about the different types of transformation scenarios available in Oxygen XML Developer plugin see the Defining a New Transformation Scenario section.
- **Storage** - Displays where a transformation scenario is stored (the Show Storage option must be enabled.)

To sort each column you can left-click its header. The contextual menu of each header allows you to do the following:

- **Show Type** - Use this option to display the transformation type of each scenario.
- **Show Storage** - Use this option to display the storage location of the scenarios.
- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.
- **Ungroup all** - Select this option to ungroup all the scenarios.
- **Reset Layout** - Select this option to restore the default settings of the layout.

The bottom section of the dialog box contains the following actions:

- **Association follows selection** - Enable this check-box to automatically associate selected transformation scenarios with the current document. This option can also be used for multiple selections.

  **Note:** When this option is enabled, the Association column is hidden.

- **New** - This button allows you to create a new transformation scenario, depending upon its type.
- **Edit** - This button opens the Edit Scenario dialog box that allows you to configure the options of the transformations scenario.

  **Note:** If you try to edit a transformation scenario associated with a defined document type, Oxygen XML Developer plugin displays a warning message to inform you that this is not possible and gives you the option to create a duplicate transformation scenario to edit instead.
• **Duplicate** - Use this button to create a *duplicate transformation scenario*.
• **Remove** - Use this button to remove transformation scenarios.

[Note: Removing scenarios associated with a defined document type is not allowed.]

The **Edit, Duplicate, Remove, Import scenarios,** and **Export selected scenarios** actions are also available in the contextual menu of the transformation scenarios listed in the middle section of the dialog box.

### Duplicating a Transformation Scenario

Use the following procedure to duplicate a transformation scenario. This is useful for creating a scenario that is similar to an existing one.

1. Open the **Configure Transformation** dialog by using the **Configure Transformation Scenario(s)** (`Alt Shift T, C (Command Alt T, C on OS X)`) action from the **Transformation** toolbar or the **XML** menu.
2. Create a copy of a scenario by selecting the scenario and clicking the **Duplicate** button.
3. Enter a new name in the **Name** field.
   a) You can choose to save the scenarios at project level by selecting the **Project Options** setting.
4. Click **OK** to save the scenario.

### Editing a Transformation Scenario

Editing a transformation scenario is useful if you need to configure some of its parameters.

Oxygen XML Developer plugin allows you to configure existing transformation scenarios by using one of the following methods:

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Then select the scenario and click the **Edit** button.
- Use the **Configure Transformation Scenario(s)** (`Alt Shift T, C (Command Alt T, C on OS X)`) action from the **Transformation** toolbar or the **XML** menu. Then select the scenario and click the **Edit** button.
- Use the **Apply Transformation Scenario(s)** (`Alt Shift T, T (Command Alt T, T on OS X)`) action from the **Transformation** toolbar or the **XML** menu. Then select the scenario and click the **Edit** button.

[Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.]

You can edit transformation scenarios that are defined at project level only. To edit a transformation scenario that is associated with a defined document type, duplicate it and edit the duplicated scenario.

### Apply Batch Transformations

A transformation action can be applied on a batch of selected files *from the contextual menu of the Project view* without having to open the files involved in the transformation. You can apply the same scenario to a batch of files or multiple scenarios to a single file or batch of files.

1. (Optional, but recommended) Organize the files you want to transform in logical folders.
   a) Create a logical folder in the **Project** view by using the **New > Logical Folder...** action from the contextual menu of the root file.
   b) Add files you want to transform to the logical folder by using the **Add Files...** or **Add Edited File** actions from the contextual menu of the logical folder.
Note: You can skip this step if the files are already in a dedicated folder that does not include any additional files or folders. You can also manually select the individual files in the Project view each time you want to transform them, but this can be tedious.

2. Right-click on the newly created logical folder and select Transform > Configure Transformation Scenario(s)... to select one or more transformation scenarios to be applied on all the files in the logical folder.

Note: These types of transformation scenarios must be configured with the current file (${cf}) or current file URL (${currentFileURL}) editor variables for the input file. This ensures that each file becomes the current file when its turn arrives in the batch transformation process. Edit the transformation scenario to make sure the appropriate editor variable is assigned for the input file. For example, for a DocBook PDF transformation make sure the XML URL input box is set to the ${currentFileURL} editor variable. For a DITA PDF transformation make sure the args.input parameter is set to the ${cf} editor variable.

3. Now that logical folder has been associated with one or more transformation scenarios, whenever you want to apply the same batch transformation you can select Transform > Transform with... from the contextual menu and the same previously associated scenario(s) will be applied.

4. If you want a different type of transformation to be applied to each file inside the logical folder, associate individual scenarios for each file and select Transform > Apply Transformation Scenario(s) from the contextual menu of the logical folder.

Built-in Transformation Scenarios

Oxygen XML Developer plugin included preconfigured built-in transformation scenarios that are used for common transformations. To obtain the desired output, use the Apply Transformation Scenario(s) (Alt Shift T, T (Command Alt T, T on OS X)) action from the Transformation toolbar or the XML menu and choose one of the built-in scenarios for the current document.

You can use the Apply Transformation Scenario(s) action even if the current document is not associated with a transformation scenario.

If the document contains an xml-stylesheet processing instruction that refers to an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Developer plugin prompts you to associate the document with a built-in transformation scenario.

The default transformation scenario is suggested based on the processing instruction from the edited document. The XSL URL field of the default transformation scenario contains the URL from the href attribute of the processing instruction. By default, the Use xml-stylesheet declaration check-box is enabled, Saxon is used as the transformation engine, and no FO processing is performed. The result of the transformation is store in a file with the same URL as the edited document, but the extension is changed to html. The name and path are preserved because the output file name is specified with the help of two editor variables: ${cfd} and ${cfn}.

Transformation Scenarios View

You can manage the transformation scenarios by using the Transformation Scenarios view. To open this view, go to Window > Show View > Transformation Scenarios.
Figure 172: The Scenarios view

The following options are available in the contextual menu of the Transformation Scenarios view:

- **Apply selected scenarios**
  Select this option to run the current transformation scenario.

- **Debug selected scenario**
  Select this option to switch to the Debugger perspective and initialize it with the parameters from the scenario (the XML, XSLT, or XQuery input, the transformation engine, the XSLT parameters).

- **Duplicate**
  Adds a new scenario to the list that is a duplicate of the current scenario. It is useful for creating a scenario that is similar to an existing one.

- **Edit**
  Opens the dialog for editing the parameters of a transformation scenario.

- **Remove**
  Removes the current scenario from the list. This action is also available by using the **Delete** key.

- **Import scenarios**
  This option opens the Import scenarios dialog that allows you to select the scenarios file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:
  - Keep or replace the existing scenario.
  - Keep both scenarios.
Export selected scenarios

Use this option to export transformation and validation scenarios individually. Oxygen XML Developer plugin creates a scenarios file that contains the scenarios that you export.

Along with the options available in the contextual menu, the Transformation Scenarios view toolbar contains a New drop-down button that contains a list of the scenarios you can create. Oxygen XML Developer plugin determines the most appropriate scenarios for the current type of file and displays them at the beginning of the list, followed by the rest of the scenarios.

The Settings drop-down menu allows you to configure the following options:

- **Show all scenarios** - Select this option to display all the available scenarios, regardless of the document they are associated with.
- **Show only the scenarios available for the editor** - Select this option to only display the scenarios that Oxygen XML Developer plugin can execute for the current document type.
- **Show associated scenarios** - Select this option to only display the scenarios associated with the document you are editing.
- **Import scenarios** - This option opens the Import scenarios dialog box that allows you to select the scenarios file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:
  - Keep or replace the existing scenario.
  - Keep both scenarios.

  Note: When you keep both scenarios, Oxygen XML Developer plugin adds imported to the name of the imported scenario.

- **Export selected scenarios** - Use this option to export transformation and validation scenarios individually. Oxygen XML Developer plugin creates a scenarios file that contains the scenarios that you export.
  - **Show Type** - Use this option to display the transformation type of each scenario.
  - **Show Storage** - Use this option to display the storage location of the scenarios.
  - **Group by Type** - Select this option to group the scenarios by their type.
  - **Group by Storage** - Select this option to group the scenarios by their storage location.
- **Ungroup all** - Select this option to ungroup all the scenarios.
- **Reset Layout** - Select this option to restore the default settings of the layout.

Oxygen XML Developer plugin supports multiple scenarios association. To associate multiple scenarios with a document, enable the check-boxes in front of each scenario. You can also associate multiple scenarios with a document from the Configure Transformation Scenario(s) or Configure Validation Scenario(s) dialogs.

The Transformation Scenarios presents both global scenarios and project scenarios. By default, Oxygen XML Developer plugin presents the items in the Transformation Scenarios in the following order: scenarios matching the current framework, scenarios matching the current project, scenarios matching other frameworks. You can group the scenarios depending on the columns in the Transformation Scenarios view. Right click the name of a column to choose how to group the scenarios. The following grouping options are available:

- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.

**Debugging PDF Transformations**

To debug a DITA PDF transformation scenario using the XSLT Debugger follow these steps:
1. **Open the Preferences dialog box**, go to XML > XML Catalog, click **Add**, and select the file located at `{OXYGEN_DIR}/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/catalog.xml`.

2. Open the map in the **DITA Maps Manager** and create a **DITA Map PDF** transformation scenario.

3. Edit the scenario, go to the **Parameters** tab and change the value of the `clean.temp` parameter to **no**.

4. Run the transformation scenario.

5. Open the `stage1.xml` file located in the temporary directory and **format and indent** it.

6. Create a transformation scenario for this XML file by associating the `topic2fo_shell_fop.xsl` stylesheet located at `{OXYGEN_DIR}/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl`. If you are specifically using the RenderX XEP or Antenna House FO processors to build the PDF output, you should use the XSL stylesheets `topic2fo_shell_xep.xsl` or `topic2fo_shell_axf.xsl` located in the same folder.

7. In the transformation scenario edit the XSLT Processor combo box choose the Saxon EE XSLT processor (the same processor used when the DITA OT transformation is executed).

8. In the transformation scenario edit the **Parameters** list and set the parameter `locale` with the value `en_GB` and the parameter `customizationDir.url` to point either to your customization directory or to the default DITA OT customization directory. Its value should have an URL syntax like: `file://c:/path/to/{OXYGEN_DIR}/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg`. The validation in schema aware transformations is done according to the W3C XML Schema 1.0 or 1.1. This can be **configured in Preferences**.

9. Debug the transformation scenario.

### XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Developer plugin.

#### Supported XSLT Processors

Oxygen XML Developer plugin includes the following XSLT processors:

- **Xalan 2.7.1 - Xalan-Java** is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.

- **Saxon 6.5.5 - Saxon 6.5.5** is an XSLT processor that implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.

- **Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE) - Saxon-HE/PE** implements the basic conformance level for XSLT 2.0 / 3.0 and XQuery 1.0. The term _basic XSLT 2.0 / 3.0 processor_ is defined in the draft XSLT 2.0 / 3.0 specifications. It is a conformance level that requires support for all features of the language other than those that involve schema processing. The HE product remains open source, but removes some of the more advanced features that are present in Saxon-PE.

- **Saxon 9.6.0.5 Enterprise Edition (EE) - Saxon EE** is the schema-aware edition of Saxon and it is one of the built-in processors included in Oxygen XML Developer plugin. Saxon EE includes an XML Schema processor, and schema-aware XSLT, XQuery, and XPath processors.

The validation in schema aware transformations is done according to the W3C XML Schema 1.0 or 1.1. This can be **configured in Preferences**.

**Note:** Oxygen XML Developer plugin implements a Saxon framework that allows you to create Saxon configuration files. Two templates are available: **Saxon collection catalog** and **Saxon configuration**. Both of these templates support content completion, element annotation, and attribute annotation.

**Note:** Saxon can use the **ICU-J localization library** (`saxon9-icu.jar`) to add support for sorting and date/number formatting in a wide variety of languages. This library is not included in the Oxygen XML Developer plugin installation kit. However, Saxon will use the default collation and localization support available in the currently used JRE. To enable this capability follow these steps:
2. Unpack the downloaded archive.
3. Copy the saxon9-icu.jar file to oxygen/lib directory.
4. Re-start the application.

- **Saxon-CE (Client Edition)** is Saxonica's implementation of XSLT 2.0 for use on web browsers. Oxygen XML Developer plugin provides support for editing stylesheets that contain Saxon-CE extension functions and instructions. This support improves the validation, content completion, and syntax highlighting.

  Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about executing such a stylesheet on Saxonica's website.

  Note: A specific template, named **Saxon-CE stylesheets**, is available in the New From Templates wizard.

- **Xsltproc (libxslt)** - Libxslt is the XSLT C library developed for the Gnome project. Libxslt is based on libxml2, the XML C library developed for the Gnome project. It also implements most of the EXSLT set of processor-portable extensions, functions, and some of Saxon's evaluate and expression extensions. The libxml2 version included in Oxygen XML Developer plugin is 2.7.6 and the Libxslt version is 1.1.26.

  Oxygen XML Developer plugin uses Libxslt through its command line tool (Xsltproc). The XSLT processor is included in the distribution kit of the stand-alone version for Windows and Mac OS X. Since there are differences between various Linux distributions, on Linux you must install Libxslt on your machine as a separate application and set the PATH variable to contain the Xsltproc executable.

  If you do not have the Libxslt library already installed, you should copy the following files from Oxygen XML Developer plugin stand-alone installation directory to the root of the com.oxygenxml.editor_17.0 plugin:

  - on Windows: xsltproc.exe, zlib1.dll, libxslt.dll, libxml2.dll, libxslt.dll, iconv.dll
  - on Linux: xsltproc, libexslt.so.0, libxslt.so.1, libxml2.so.2
  - on Mac OS X: xsltproc.mac, libexslt, libxslt, libxml

  Note: The Xsltproc processor can be configured from the **XSLTPROC options page**.

  Caution: Known problem: file paths containing spaces are not handled correctly in the LIBXML processor. For example, the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled properly by LIBXML if Oxygen XML Developer plugin is installed in the default location on Windows (C:\Program Files). This is because the built-in XML catalog files are stored in the [OXYGEN_DIR]/frameworks subdirectory of the installation directory, which in this case contains at least a space character.

- **MSXML 4.0** - **MSXML 4.0** is available only on Windows platforms. It can be used for transformation and validation of XSLT stylesheets.

  Oxygen XML Developer plugin uses the Microsoft XML parser through its command line tool msxml.exe. Since msxml.exe is only a wrapper, Microsoft Core XML Services (MSXML) must be installed on the computer. Otherwise, you will get a corresponding warning. You can get the latest Microsoft XML parser from Microsoft web-site.

- **MSXML .NET** - **MSXML .NET** is available only on Windows platforms. It can be used for transformation and validation of XSLT stylesheets.

  Oxygen XML Developer plugin performs XSLT transformations and validations using .NET Framework's XSLT implementation (System.Xml.Xsl.XslTransform class) through the nxslt command line utility. The nxslt version included in Oxygen XML Developer plugin is 1.6.
You should have the .NET Framework version 1.0 already installed on your system. Otherwise, you will get the following warning: MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128.

You can get the .NET Framework version 1.0 from the Microsoft website.

- **.NET 1.0** - A transformer based on the System.Xml 1.0 library available in the .NET 1.0 and .NET 1.1 frameworks from Microsoft (http://msdn.microsoft.com/xml/). It is available only on Windows.

  You should have the .NET Framework version 1.0 or 1.1 already installed on your system. Otherwise, you will get the following warning: MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128.

  You can get the .NET Framework version 1.0 from the Microsoft website.

- **.NET 2.0** - A transformer based on the System.Xml 2.0 library available in the .NET 2.0 framework from Microsoft. It is available only on Windows.

  You should have the .NET Framework version 2.0 already installed on your system. Otherwise, you will get the following warning: MSXML.NET requires .NET Framework version 2.0 to be installed. Exit code: 128.

  You can get the .NET Framework version 2.0 from the Microsoft website.

### Configuring Custom XSLT Processors

You can configure and run XSLT and XQuery transformations with processors other than the ones which come with the Oxygen XML Developer plugin distribution.

**Note:**

The output messages of a custom processor are displayed in an output view at the bottom of the application window. If an output message follows the format of an Oxygen XML Developer plugin linked message, then a click on the message in the output view highlights the location of the message in an editor panel containing the file referenced in the message.

### Configuring the XSLT Processor Extensions Paths

The Xalan and Saxon processors support the use of extension elements and extension functions. Unlike a literal result element, which the stylesheet simply transfers to the result tree, an extension element performs an action. The extension is usually used because the XSLT stylesheet fails in providing adequate functions to the user for accomplishing a more complex task.

Samples on how to use extensions can be found at:

- for Saxon 6.5.5 - http://saxon.sourceforge.net/saxon6.5.5/extensions.html

To set an XSLT processor extension (a directory or a jar file), use the Extensions button in the Edit scenario dialog box.

**Note:** The old way of setting an extension (using the parameter -Dcom.oxygenxml.additional.classpath) was deprecated, and instead you should use the extension mechanism of the XSLT transformation scenario.

### XSL-FO Processors

This section explains how to apply XSL-FO processors when transforming XML documents to various output formats in Oxygen XML Developer plugin.

#### The Built-in XSL-FO Processor

The Oxygen XML Developer plugin installation package is distributed with the Apache FOP that is a Formatting Objects processor for rendering your XML documents to PDF. FOP is a print and output independent formatter driven by XSL
Formatting Objects. **FO** is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

To include PNG images in the final PDF document you need the **JIMI** or **JAI** libraries. For PDF images you need the **fop-pdf-images** library. These libraries are not bundled with Oxygen XML Developer plugin but using them is very easy. You need to download them and create an external FO processor based on the built-in FOP libraries and the extension library. The external FO processor created in Preferences will have a command line like:

```
java -cp "${oxygenInstallDir}/lib/xercesImpl.jar:
${oxygenInstallDir}/lib/fop.jar:${oxygenInstallDir}/lib/
avalon-framework-4.2.0.jar:
${oxygenInstallDir}/lib/batik-all-1.7.jar:${oxygenInstallDir}/lib/
commons-io-1.3.1.jar:
${oxygenInstallDir}/lib/xmlgraphics-commons-1.3.1.jar:
${oxygenInstallDir}/lib/commons-logging-1.0.4.jar:
${oxygenInstallDir}/lib/saxon9ee.jar:${oxygenInstallDir}/lib/
saxon9-dom.jar:
${oxygenInstallDir}/lib/xml/seraliser.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 OS X version of the **JAI** library can be downloaded from [http://www.apple.com/downloads/macosx/apple/java3dandjavaadvancedimagingupdate.html](http://www.apple.com/downloads/macosx/apple/java3dandjavaadvancedimagingupdate.html). In order to use it, install the downloaded package.

Other FO processors can be configured in the Preferences dialog box.

### Add a Font to the Built-in FOP - The Simple Version

If the font that must be set to Apache FOP is one of the fonts that are installed in the operating system you should follow the next steps for creating and setting a FOP configuration file that looks for the font that it needs in the system fonts. It is a simplified version of the [procedure for setting a custom font in Apache FOP](http://oxygenxmldeveloper.sourceforge.net/fop.html).

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.

   ```xml
   <fop version="1.0">
   <renderers>
       <renderer mime="application/pdf">
       <fonts>
           <auto-detect/>
       </fonts>
   </renderer>
   </renderers>
   </fop>
   ```

   b) **Open the Preferences dialog box**, go to **XML > XSLT/FO/XQuery > FO Processors**, and enter the path of the FOP configuration file in the Configuration file text field.

2. Set the font on the document content.
   This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

   - For DocBook documents you can start with the predefined scenario called **DocBook PDF**, edit the XSLT parameters and set the font name (in our example the font family name is Arial Unicode MS) to the parameters body.font.family and title.font.family.
   - For TEI documents you can start with the predefined scenario called **TEI PDF**, edit the XSLT parameters and set the font name (in our example Arial Unicode MS) to the parameters bodyFont and sansFont.
   - For DITA transformations to PDF using DITA-OT you should modify the following two files:
• [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml
  - the font-face element included in each element physical-font having the attribute char-set="default" must contain the name of the font (Arial Unicode MS in our example)
• [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf
  - an element auto-detect must be inserted in the element fonts which is inside the element renderer having the attribute mime="application/pdf":

```xml
<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.

1. **Important:** If this special font is installed in the operating system, there is a simple way of telling FOP to look for it. See the simplified procedure for adding a font to FOP.

   1. **Locate the font.**

      First, find out the name of a font that has the glyphs for the special characters you used. One font that covers most characters, including Japanese, Cyrillic, and Greek, is Arial Unicode MS.

      On Windows the fonts are located into the `C:\Windows\Fonts` directory. On Mac, they are placed in `/Library/Fonts`. To install a new font on your system, is enough to copy it in the `Fonts` directory.

   2. **Generate a font metrics file from the font file.**

      a) Open a terminal.
      b) Change the working directory to the Oxygen XML Developer plugin install directory.
      c) Create the following script file in the Oxygen XML Developer plugin installation directory.

      For OS X and Linux create a file `ttfConvert.sh`:

      ```bash
      #!/bin/sh
      export LIB=lib
      export CP=$LIB/fop.jar
      export CP=$CP:$LIB/avalon-framework-4.2.0.jar
      export CP=$CP:$LIB/xercesImpl.jar
      export CP=$CP:$LIB/commons-logging-1.1.1.jar
      export CP=$CP:$LIB/commons-io-1.3.1.jar
      export CP=$CP:$LIB/xmlgraphics-commons-1.5.jar
      export CP=$CP:$LIB/xml-apis.jar
      export CMD="java -cp "$CP" org.apache.fop.fonts.apps.TTFReader"
      export FONT_DIR=.
      $CMD $FONT_DIR/Arialuni.ttf Arialuni.xml
      ```

      For Windows create a file `ttfConvert.bat`:

      ```bash
      @echo off
      set LIB=lib
      set CP=\%LIB\%\fop.jar
      set CP=\%CP%;\%LIB\%\avalon-framework-4.2.0.jar
      set CP=\%CP%;\%LIB\%\xercesImpl.jar
      set CP=\%CP%;\%LIB\%\commons-logging-1.1.1.jar
      set CP=\%CP%;\%LIB\%\commons-io-1.3.1.jar
      set CP=\%CP%;\%LIB\%\xmlgraphics-commons-1.5.jar
      set CP=\%CP%;\%LIB\%\xml-apis.jar
      set CMD=java -cp "%CP" org.apache.fop.fonts.apps.TTFReader
      set FONT_DIR=C:\Windows\Fonts
      %CMD% %FONT_DIR%\Arialuni.ttf Arialuni.xml
      ```
The paths specified in the file are relative to the Oxygen XML Developer plugin installation directory. If you decide to create it in another directory, change the file paths accordingly.

The `FONT_DIR` can be different on your system. Check that it points to the correct font directory. If the Java executable is not in the `PATH`, specify the full path of the executable.

If the font has bold and italic variants, convert them too by adding two more lines to the script file:

- for OS X and Linux:
  ```bash
  $CMD $FONT_DIR/Arialuni-Bold.ttf Arialuni-Bold.xml
  $CMD $FONT_DIR/Arialuni-Italic.ttf Arialuni-Italic.xml
  ```

- for Windows:
  ```bash
  %CMD% %FONT_DIR%\Arialuni-Bold.ttf Arialuni-Bold.xml
  %CMD% %FONT_DIR%\Arialuni-Italic.ttf Arialuni-Italic.xml
  ```

d) Execute the script.

On Linux and OS X, execute the command `sh ttfConvert.sh` from the command line. On Windows, run the command `ttfConvert.bat` from the command line or double click on the file `ttfConvert.bat`.

3. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)
   a) Create a FOP configuration file that specifies the font metrics file for your font.

```xml
<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>
        <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>
    </renderer>
  </renderers>
</fop>
```

The `embed-url` attribute points to the font file to be embedded. Specify it using the URL convention. The `metrics-url` attribute points to the font metrics file with a path relative to the `base` element. The triplet refers to the unique combination of name, weight, and style (italic) for each variation of the font. In our case is just one triplet, but if the font had variants, you would have to specify one for each variant. Here is an example for Arial Unicode if it had italic and bold variants:

```xml
<fop version="1.0">
  ...
  <font metrics-url="Arialuni.xml" kerning="yes"
         embed-url="file:/Library/Fonts/Arialuni.ttf">
    <font-triplet name="Arialuni" style="normal" weight="normal"/>
  </font>
  ...
</fop>
```
More details about the FOP configuration file are available on the FOP website.

b) **Open the Preferences dialog box**, go to XML > XSLT/FO/XQuery > FO Processors, and enter the path of the FOP configuration file in the **Configuration file** text field.

4. Set the font on the document content.
   
   This is usually done with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.
   
   For DocBook documents, you can start with the predefined scenario called **DocBook PDF**, edit the XSLT parameters, and set the font name (in our example Arialuni) to the parameters body.font.family and title.font.family.
   
   For TEI documents, you can start with the predefined scenario called **TEI PDF**, edit the XSLT parameters, and set the font name (in our example Arialuni) to the parameters bodyFont and sansFont.
   
   For DITA to PDF transformations using DITA-OT modify the following two files:

   - [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml - the font-face element included in each element physical-font having the attribute char-set="default" must contain the name of the font (Arialuni in our example)
   - [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf - an element font must be inserted in the element fonts which is inside the element renderer having the attribute mime="application/pdf":

   ```xml
   <renderer mime="application/pdf">
     <...>
     <font metrics-url="Arialuni.xml" kerning="yes" embed-url="file:/Library/Fonts/Arialuni.ttf">
       <font-triplet name="Arialuni" style="normal" weight="normal"/>
     </font>
     </fonts>
   </renderer>
   ```

**Adding Libraries to the Built-in FOP**

You can extend the functionality of the built-in FO processor by dropping additional libraries in the [OXYGEN_DIR]/lib/fop directory.

**Hyphenation**

To add support for hyphenation:

1. download the pre-compiled JAR from OFFO;
2. place the JAR in [OXYGEN_DIR]/lib/fop;
3. restart the Oxygen XML Developer plugin.

**Output Formats**

Oxygen XML Developer plugin allows you to use transformation scenarios to publish XML content in various output formats (such as WebHelp, PDF, CHM, EPUB, JavaHelp, Eclipse Help, XHTML, etc.)

For transformations that are not included in your installed version of Oxygen XML Developer plugin, simply install the tool chain required to perform the specific transformation and process the files in accordance with the processor instructions. A multitude of target formats are possible. The basic condition for transformation to any format is that your source document is well-formed.

Note: You need to use the appropriate stylesheet according to the source definition and the desired output. For example, if you want to transform into an HTML format using a DocBook stylesheet, your source XML document should conform with the DocBook DTD.

For more information, see the **Transformation Scenarios** on page 308 section.
WebHelp Output

Oxygen XML Developer plugin allows you to obtain WebHelp output from DocBook and DITA documents. This section contains information about the WebHelp system, its variants, and ways to customize it to better fit your specific needs.

WebHelp System Description

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Layout

The layout of the WebHelp system is comprised of two parts:

- The left section that contains separate tabs for Content, Search, and Index.

  Note: If your documents contain no indexterm elements, the Index tab is not generated.

  Note: You can enhance the appearance of the selected item in the Table of Contents. See the Customizing WebHelp chapter for more details.

- The right section where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper-right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

  Note: You can edit the args.hide.parent.link parameter to hide the Parent, Next, and Previous links.

You can use the Collapse all button that is displayed in the Content tab to collapse all the topics presented in the Table of Contents.

The top-right corner of the page contains the following options:

- With Frames - Displays the output using HTML frames to render two separate sections (a section that displays the Table of Contents in the left side and a section that displays the content of a topic in the right side).
- Print this page - Opens a dialog with various printing options and a print preview.
The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on the following:

- The number of keywords found in a single page (the higher the number, the better).
- The context (for example, a word found in a title scores better than a word found in unformatted text). The search ranking order, sorted by relevance is as follows:
  - The search phrase is included in a meta keyword
  - The search phrase is in the title of the page
  - The search phrase is in bold text in a paragraph
  - The search phrase is in normal text in a paragraph
Figure 174: WebHelp Search with Stemming Enabled

Rules that are applied during a search include:

- The space character separates keywords (an expression such as *grow flowers* counts as two separate keywords: *grow* and *flowers*).
- Do not use quotes to perform an exact search for multiple word expressions (an expression such as "*grow flowers*", returns no results since it searches for two separate words).
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page (for example, content inside `keywords` elements weighs twice as much as content inside an `H1` HTML element).
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot ("."), characters count as a single word.
- Always search for words containing three or more characters (shorter words, such as *to* or *of* are ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

This output format is compatible with the following browsers:

- Internet Explorer (8 or newer)
- Chrome
- Firefox
- Safari
- Opera

**Important:** Due to some security restrictions in Google Chrome, WebHelp pages loaded from the local system (through URLs of the `file:///`... format) may not work properly. We recommend that you load WebHelp pages in Google Chrome only from a web server (with a URL such as
Warning: Due to some restrictions in web browsers in regards to JavaScript code, the frameless version (index.html start page) of the WebHelp system should only be loaded from a web server (with a URL such as http://your.server.com/webhelp/index.html or http://localhost/web_pages/index.html). When loading WebHelp pages from the local file system, the frameset version (index_frames.html start page) of the WebHelp system should be used instead (file:///...).

WebHelp with Feedback System Description

WebHelp with Feedback is a form of online help system that consists of a series of web pages (XHTML format). Its advantages include platform independence, continuous content update, and a feedback mechanism that allows your authors and audience to interact with one another.

Layout

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp and the left section is the same. However, the bottom of the right section contains a comments bar. Select Log in from this bar to authenticate as a user for the WebHelp system. If you do not have a user name, complete the fields in the dialog box to create a user. Under the comments bar, you can click the Add New Comment button to add a comment, regardless of whether or not you are logged in.

Note: You can enhance the appearance of the selected item in the Table of Contents. See the Customizing WebHelp chapter for more details.

Figure 175: The Layout of the Feedback-Enabled WebHelp System
After you log in, your name and user name are displayed in the Comments bar together with the Log of and Edit buttons. Click the Edit button to open the User Profile dialog. In this dialog you can customize the following options:

- **Your Name** - you can use this field to edit the initial name that you used to create your user profile.
- **Your e-mail address** - you can use this field to edit the initial e-mail address that you used to create your profile.
- **When to receive an e-mail:**
  - When a comment is left on a page that you commented on.
  - When a comment is left on any topic in the Help system.
  - When a reply is left to one of my comments.
- **New Password** - allows you to enter a new password for your user account.

Note: The Current Password field from the top of the User Profile is mandatory in case you want to save the changes you make.

**Search Tab**

The Search tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on the following:

- The number of keywords found in a single page (the higher the number, the better).
- The context (for example, a word found in a title scores better than a word found in unformatted text). The search ranking order, sorted by relevance is as follows:
  - The search phrase is included in a meta keyword
  - The search phrase is in the title of the page
  - The search phrase is in bold text in a paragraph
  - The search phrase is in normal text in a paragraph
Figure 176: WebHelp Search with Stemming Enabled

Rules that are applied during a search include:

- The space character separates keywords (an expression such as *grow flowers* counts as two separate keywords: *grow* and *flowers*).
- Do not use quotes to perform an exact search for multiple word expressions (an expression such as "*grow flowers*", returns no results since it searches for two separate words).
- *indexterm* and *keywords* DITA elements are an effective way to increase the ranking of a page (for example, content inside *keywords* elements weighs twice as much as content inside an *H1* HTML element).
- Words composed by merging two or more words with colon (":") minus ("-"), underline ("_"), or dot ("." ) characters count as a single word.
- Always search for words containing three or more characters (shorter words, such as *to* or *of* are ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

This output format is compatible with the following browsers:

- Internet Explorer (8 or newer)
- Chrome
- Firefox
- Safari
- Opera
Deployment of the WebHelp With Feedback System

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Developer plugin requires the following system components:

- Apache Web Server running
- MySQL server running
- PHP Version 5.1.6 or later
- PHP MySQL Support

Oxygen XML WebHelp system supports the following browsers: IE7+, Chrome 19+, Firefox 11+, Safari 5+, Opera 11+

Installation Instructions

Note: These instructions were written for XAMPP 1.7.7 with PHP 5.3.8 and for phpMyAdmin 3.4.5. Later versions of these packages may change the location or name of some options, however the following installation steps should remain valid and basically the same.

In case you have a web server configured with PHP, MySQL, you can deploy the WebHelp output directly. Otherwise, install XAMPP. XAMPP is a free and open source cross-platform web server solution stack package. It consists mainly of the Apache HTTP Server, MySQL database, and interpreters for scripts written in PHP.

Install XAMPP

1. Go to https://www.apachefriends.org/download.html and download XAMPP, for instance for a Windows system.
2. Install it in C:\xampp.
3. From the XAMPP control panel, start MySQL, and then Apache.
4. Open http://localhost/xampp/index.php in your browser to check whether the HTTP server is working.

Create the WebHelp Feedback database

The WebHelp system needs a database to store user details and the actual feedback they provide. The following procedure creates a database for the feedback system and a MySQL user with privileges on that database. The feedback system uses these credentials to connect to the database.

Use phpMyAdmin to create a database:

1. Type localhost in your browser.
2. In the left area, select: phpMyAdmin.
3. Click Databases (in the right frame) and then create a database. You can give any name you want to your database, for example comments.
4. Create a user with connection privileges for this database. In the SQL tab, paste the following text:

```
INSERT INTO `mysql`.`user`
(`Host`, `User`, `Password`, `Select_priv`, `Insert_priv`, `Update_priv`, `Delete_priv`, `Create_priv`,
`Drop_priv`, `Reload_priv`, `Shutdown_priv`, `Process_priv`, `File_priv`, `Grant_priv`, `References_priv`, `Index_priv`, `Alter_priv`,
`Show_db_priv`, `Super_priv`, `Create_tmp_table_priv`, `Lock_tables_priv`, `Execute_priv`, `Repl_slave_priv`, `Repl_client_priv`,
`Create_view_priv`,
`Show_view_priv`, `Create_routine_priv`, `Alter_routine_priv`, `Create_user_priv`, `Event_priv`, `Trigger_priv`,
`Create_tablespace_priv`, `sql_type`, `max_questions`, `max_updates`, `max_connections`, `max_user_connections`, `plugin`,
`authentication_string`) VALUES ('localhost', 'user_name', PASSWORD('user_password'),
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N',
'0', '0', '', '');
```
5. Change the user_name and the user_password values.
6. Under localhost in the right frame click Privileges and then at the bottom of the page click the reload the privileges link.

Deploying the WebHelp output

To deploy the WebHelp output, follow these steps:

1. Locate the directory of the HTML documents. Open http://localhost/xampp/phpinfo.php in your browser and see the value of the DOCUMENT_ROOT variable. In case you installed XAMPP in C:\xampp, the value of DOCUMENT_ROOT is C:/xampp/htdocs.
2. Copy the transformation output folder in the DOCUMENT_ROOT.
3. Rename it to a relevant name, for example, webhelp_1.
   - Verify that the prerequisites are met.
   - Press Start Installation.
   - Configure the Deployment Settings section. Default values are provided, but you should adjust them as needed.
   - Configure the MySql Database Connection Settings section. Use the details from the Create the WebHelp Feedback database section to fill-in the appropriate text boxes.

   **Warning:** Checking the Create new database structure option will overwrite any existing data in the selected database, if it already exists.

   • If you are using a domain (such as OpenLDAP or Active Directory) to manage users in your organization, check the Enable LDAP Autentification option. This will allow you to configure the LDAP server, which will provide information and credentials for users who will access the WebHelp system. Also, this will allow you to choose which of the domain users will have administrator privileges.
   • If the Create new database structure option is checked, the Create WebHelp Administrator Account section becomes available. Here you can set the administrator account data. The administrator is able to moderate new posts and manage WebHelp users.

   The same database can be used to store comments for different WebHelp deployments. If a topic is available in more than one WebHelp deployments and there are comments associated with it, you can choose to display the comments in all deployments that share the database. To do this, enable the Display comments from other products option. In the Display comments from section a list with the deployments sharing the same database is displayed. Select the deployments allowed to share common feedback.

   **Note:** You can restrict the displayed comments of a product depending on its version. In case you have two products that use the same database and you restrict one of them to display comments starting from a certain version, the comments of the other product are also displayed from the specified version onwards.

   • Press Next Step.
   • Remove the installation folder from your web server.
   • Click the link pointing to the index of the documentation, or visit: http://localhost/webhelp_1/.

To test your system, create a user and post a comment. Check if the notification emails are delivered to your inbox.

**Note:** To read debug messages generated by the system:

1. Enable JScript logging by doing one of the following:
   • Open the log.js file, locate the var log= new Log(Level.NONE); line, and change the logging level to: Level.INFO, Level.DEBUG, Level.WARN, or Level.ERROR.
   • Append ?log=true to the WebHelp URL.
2. Inspect the PHP and Apache server log files.
Feedback System User Management

Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. To access the administrative page, select Admin Panel from the Comments bar.

Figure 177: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp system. You can also view the details of each user and search through these details using the Search User Information filter.

The upper part of the page contains the following actions:

- **Delete Orphan Comments** - deletes comments associated with topics that are no longer available
- **Delete Pending Users** - deletes all unconfirmed users that registered more than a week ago
- **View All Posts** - allows you to view all posts associated with a product and version
- **Export Comments** - allows you to export in XML format all posts associated with a product and version
- **Set Version** - use this action to display comments starting from a particular version

To edit the details of a user, click the corresponding row. Use the Edit User dialog box to customize all the information associated with an user:

- **Name** - The user's full name
- **Level** - Use this field to modify the privilege level of the currently edited user. You can choose from:
  - **User** - regular user, able to post comments and receive e-mail notifications
  - **Moderator** - in addition to the regular User rights, this type of user has access to the Admin Panel. In the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system.
  - **Admin** - full administrative privileges. Can manage WebHelp-specific settings, users and their comments.
- **Company** - User's organization name
- **E-mail** - User's contact e-mail address. This is also the address where the WebHelp system sends notifications:
  - **WebHelp Notification** - when enabled, the user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system
  - **Reply Notification** - when enabled, the user receives notifications when comments are posted as a reply to one of his or hers comments
  - **Page Notification** - when enabled, the user receives notifications when comments are posted on a topic where he or she posted a comment
- **Date** - User registration date
- **Status** - Specifies the status of the currently edited user:
  - **Created** - the user is created but does not have any rights over the feedback-enabled WebHelp system
  - **Validated** - the user is able to use the feedback-enabled WebHelp system
  - **Suspended** - the user has no rights over the feedback-enabled WebHelp system
WebHelp Mobile System Description

To further improve its ability to create online documentation, Oxygen XML Developer plugin offers support to transform DocBook And DITA documents into Mobile WebHelp systems. This feature generates an output that works on multiple platforms (Android, iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices. All the specific touch screen gestures are supported. The functionality of the desktop WebHelp layout is preserved, is organized in an intuitive layout, and offers table of contents, search capabilities, and index navigation.

![Welcome to Docbook Support in oXygen](image)

**Figure 178: Mobile WebHelp**

Context-Sensitive WebHelp System

Context-sensitive help systems assist users by providing specific informational topics for certain components of a user interface, such as a button or window. This mechanism works based on mappings between a unique ID defined in the topic and a corresponding HTML page.

When WebHelp output is generated by Oxygen XML Developer plugin, the transformation process produces an XML mapping file called `context-help-map.xml` and copies it in the output folder. This XML file maps an ID to a corresponding HTML page like:

```xml
<map productID="oxy-webhelp" productVersion="1.1">
  <appContext helpID="annotations-view" path="topics/annotations-view.html"/>
  <appContext helpID="button-editor" path="concepts/button-editor.html"/>
  
  ...
</map>
```

where:

- **helpID** - unique ID provided by a topic from two possible sources:
  - the `resourceid` element set to it in the prolog section:

```xml
<prolog>
  <resourceid id="context-sensitive-help-system"/>
</prolog>
```

- **Note:** If you need different parts of the application (for instance, dialog boxes, views, or editing modes) to open the same contextual help topic, all of the context ID values should be included in the same DITA topic file. For example, if you need both a dialog box and a view to open the same WebHelp page, you can assign different resource ID in the same DITA topic.

```xml
<prolog>
  <resourceid id="dialog1"/>
```
• the `id` attribute set on the topic root element

**Important:** You should ensure that these defined IDs are unique in the context of the entire DITA project. If the IDs are not unique, the transformation scenario will display warning messages in the transformation console output. In this case the help system will not work properly.

• `path` - path to a corresponding WebHelp page. This path is relative to the location of the `context-help-map.xml` mapping file.
• `productId` - ID of the product for which you are writing documentation. Applicable only if you are using WebHelp with Feedback transformations.
• `productVersion` - version of the product for which you are writing documentation. Applicable only if you are using WebHelp with Feedback transformations.

There are two ways of implementing context-sensitive help in your application:

• The XML mapping file can be loaded by a PHP script on the server side. The script receives the context ID value and will look it up in the XML file.
• Invoke one of the WebHelp system files `index.html` or `index_frames.html` and pass them the `contextId` parameter with a specific value. The WebHelp system will automatically open the help page associated with the value of the `contextId` parameter.

```
The following example will open a frameless version of the WebHelp system showing the page associated with the id `dialog1ID`:

index.html?contextId=dialog1ID
```

```
The following example will open a frameset version of the WebHelp system showing the page associated with the id `view1ID`:

index_frames.html?contextId=view1ID
```

### Customizing the WebHelp Systems

This section contains various customizations that you can make to the output of your WebHelp transformation.

To change the overall appearance of the WebHelp output, you can use the visual **WebHelp Skin Builder tool**, which does not require knowledge of CSS language.

If you are familiar with CSS and coding, this section includes topics that explain how you can customize your WebHelp system, such as how to improve the appearance of the Table of Contents, add logo images in the title area, remove the navigation buttons, and add custom headers and footers.

### The WebHelp Skin Builder

The **WebHelp Skin Builder** is a simple, easy-to-use tool, specially designed to assist users to visually customize the look and feel of the WebHelp output. It is implemented as an online tool hosted on the Oxygen XML Developer plugin website and allows you to experiment with different styles and colors over an inert documentation sample.

To be able to use the **Skin Builder**, you need:

• An Internet connection and unrestricted access to Oxygen XML Developer plugin website.
• A later version web browser.
To start the **Skin Builder**, do one of the following:

- From the Oxygen XML Developer plugin in the **Skins tab**, click the **Online preview** link. In the upper section of the preview, click the **Select Skin** button, then choose **Customize Skin**.

### The Skin Builder Layout

The left side panel of the **Skin Builder** is divided into 3 sections:

- **Actions** - contains two buttons:
  - **Import** - allows you to load a CSS stylesheet and applies it over the documentation sample.
  - **Export** - saves all properties as a CSS file.

- **Settings** - contains the **Highlight selection** checkbox which helps you identify the areas affected by a particular element customization:
  - When hovering an item in the customizable elements menu, the affected sample area is highlighted with a dotted blue border.
  - When an item in the customizable elements menu is selected, the affected sample area is highlighted with a solid red border.

- **Customize** - provides a series of customizable elements organized under four main categories:
  - **Header**
  - **TOC Area**
  - **Vertical Splitter**
  - **Content**

  For each customizable element you can alter properties like background color or font face. Any alteration made in the customizable elements menu is applied in real time over the sample area.

### Create a Customization Skin

- The starting point can be either one of the predefined skins or a CSS stylesheet applied over the sample using the **Import** button.
- Use the elements in the **Customize** section to set properties that modify the skin's look. By default, all customizable elements display a single property, but you can make more properties visible if you click the **Add** button and choose from the available ones.

  **Note:** If you want to revert a setting of a particular property to its initial value, press the **Reset** button.

- When you are happy with the skin customization you have made, press the **Export** button. All settings will be saved in a CSS file.

### Apply a Customization Skin to a DITA Map to WebHelp Transformation Scenario

- Start Oxygen XML Developer plugin.
- Load the DITA Map you want to produce as a WebHelp output.
- Edit a **DITA Map to WebHelp**-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins** tab.
- Execute the transformation to obtain the WebHelp output.

### Apply a Customization Skin to a DocBook to WebHelp Transformation Scenario

- Start Oxygen XML Developer plugin.
- Load the DocBook file you want to produce as a WebHelp output.
• Edit a *DocBook to WebHelp*-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins** tab.
• In the **Parameters** tab, set the `webhelp.skin.css` parameter to point to the previously exported CSS.
• To customize the logo, use the following parameters:
  • `webhelp.logo.image` - Specifies a path to an image displayed as a logo in the left side of the output header.
  • `webhelp.logo.image.target.url` - Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

**Automating the WebHelp Output**

Oxygen XML WebHelp plugin allows you to use a command line interface script to obtain WebHelp output from DITA and DocBook documents. Note that the Oxygen XML WebHelp plugin is a standalone product with its own licensing terms and cannot be used with a Oxygen XML Developer plugin license.

The WebHelp output files created with the Oxygen XML WebHelp plugin are the same as the output files produced when you run DITA or DocBook to WebHelp transformation scenarios from within Oxygen XML Developer plugin.

When an automated process is required due to the amount of output needed, do the following:

1. Install the Oxygen XML WebHelp plugin

**Oxygen XML WebHelp Plugin for DITA**

To transform DITA documents using the Oxygen XML WebHelp plugin, first integrate the plugin with the DITA Open Toolkit. The purpose of the integration is to add to the DITA Open Toolkit the following transformation types:

• `webhelp` - the transformation that produces *WebHelp* output for desktop
• `webhelp-feedback` - the transformation that produces feedback-enabled *WebHelp* for desktop
• `webhelp-mobile` - the transformations that produces *WebHelp* output for mobile devices

**Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit**

The requirements of the Oxygen XML WebHelp plugin for the DITA Open Toolkit are:

• Java Virtual Machine 1.6 or later
• DITA Open toolkit 1.6.x, 1.7.x, 1.8, or 2.0 (Full Easy Install)
• Saxon 9.1.0.8

To integrate the Oxygen XML WebHelp plugin with the DITA Open Toolkit, follow these steps:

1. Download and install a [Java Virtual Machine](http://www.oracle.com/technetwork/java/javase/downloads/index.html) 1.6 or later.
2. Download and install **DITA Open Toolkit** 1.6.x, 1.7.x, 1.8, or 2.0.
3. Navigate to the `plugins` directory located in the installation directory of the DITA Open Toolkit.
4. Copy the `com.oxygenxml.webhelp` and `com.oxygenxml.highlight` directories inside the `plugins` directory. The `com.oxygenxml.highlight` directory add syntax highlight capabilities to your WebHelp output for codeblock sections that contain source code snippets (XML, Java, JavaScript etc.).
5. If you are using DITA-OT version 2.0, the WebHelp plugin contains a `plugin_2.x.xml` which needs to be renamed to `plugin.xml`.
6. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.
7. Go to [http://sourceforge.net/projects/saxon/files/Saxon-B/9.1.0.8/] download and unzip the `processor.saxonb9-1-0-8j.zip` file (contains the Saxon 9.1.0.8).

**Registering the Oxygen XML WebHelp Plugin**

To register the Oxygen XML WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Open the `[DITA-OT-install-dir]/plugins/com.oxygenxml.webhelp` directory and create a file called `licensekey.txt`.
2. In this file, copy your license key which you purchased for your Oxygen XML WebHelp plugin.
The WebHelp transformation process reads the Oxygen XML Developer plugin license key from this file. In case the file does not exit, or it contains an invalid license, an error message will be displayed.

Running a DITA Transformation Using the Oxygen XML WebHelp Plugin

To run a DITA to WebHelp (webhelp, webhelp-feedback, webhelp-mobile) transformation using the Oxygen XML WebHelp plugin, use:

- The `dita.bat` script file for Windows based systems.
- The `dita.sh` script file for Unix/Linux based systems.

Note: You can call these files in an automated process or from the command line.

The `dita.bat` and the `dita.sh` files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate a WebHelp system, customize them to match the paths to the JVM, DITA Open Toolkit and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- `JVM_INSTALL_DIR` - specifies the path to the Java Virtual Machine installation directory on your disk.
- `DITA_OT_INSTALL_DIR` - specifies the path to DITA Open Toolkit installation directory on your disk.
- `SAXON_9_DIR` - specifies the path to the directory on your disk where you unzipped the Saxon 9 archive files.
- `TRANSTYPE` - specifies the type of the transformation you want to execute. You can set it to `webhelp`, `webhelp-feedback` and `webhelp-mobile`.
- `DITA_MAP_BASE_DIR` - specifies the path to the directory where the input DITA Map file is located.
- `DITAMAP_FILE` - specifies the input DITA Map file.
- `DITAVAL_FILE` - specifies the .ditaval input filter that the transformation process applies to the input DITA Map file.
- `DITAVAL_DIR` - specifies the path to the directory where the .ditaval file is located.
- `Doutput.dir` - specifies the output directory of the transformation.

The `-Dargs.filter` and the `-Ddita.input.valfile` parameters are optional.

Additional Oxygen XML WebHelp Plugin Parameters for DITA

You are able to append the following parameters to the command line that runs the transformation:

- `-Dwebhelp.copyright` - the copyright note that is added in the footer of the Table of Contents frame;
- `-Dwebhelp.footer.file` - specifies the location of a well-formed XHTML file containing your custom footer for the document body. Corresponds to the WEBHELP/footer_FILE XSLT parameter. The fragment must be a well-formed XHTML, with a single root element. As a common practice, place all the content into a `<div>` element;
- `-Dwebhelp.footer.include` - specifies whether the content of file set in the `-Dwebhelp.footer.file` is used as footer in the WebHelp pages. Its values can be `yes`, or `no`;
- `-Dwebhelp.product.id` - the value of this parameter is a text string, that the webhelp-feedback transformation requires. It represents a short name of the documentation target (product). All the user comments that are posted in the WebHelp output pages and are added in the comments database are bound to this product ID;
- `-Dwebhelp.product.version` - the value of this parameter is a text string, that the webhelp-feedback transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.

Note: You can deploy documentation for multiple products on the same server.

- `-Dwebhelp.product.version` - the value of this parameter is a text string, that the webhelp-feedback transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.

Note: Multiple documentation versions can be deployed on the same server.

In case you need to further customize the transformation process, you are able to append other DITA-OT parameters as well. Any parameter that you want to append must follow the `-D` model of the above parameters. For example, to append the `args.hdr` parameter, use:

`-Dargs.hdr=[HEADER_FILE_DIR]`
where \([\text{HEADER\_FILE\_DIR}]\) is the location of the directory that contains the header file.

Database Configuration for DITA WebHelp with Feedback

If you run the `webhelp-feedback` transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at \([\text{DITA\_MAP\_BASE\_DIR}]\)/out/[\TRANSFORM\_TYPE]/oxygen-webhelp/resources. The `installation.html` file is created by the transformation process.

**Oxygen XML WebHelp Plugin for DocBook**

To transform DocBook documents using the Oxygen XML WebHelp plugin, first integrate the plugin with the DocBook XSL distribution. The purpose of the integration is to add to the DocBook XSL distribution the following transformation types:

- `webhelp` - the transformation that produces `WebHelp` output for desktop
- `webhelp-feedback` - the transformation that produces feedback-enabled `WebHelp` for desktop
- `webhelp-mobile` - the transformations that produce `WebHelp` output for mobile devices

Integrating the Oxygen XML WebHelp Plugin with the DocBook XSL Distribution

The WebHelp plugin transformations run as an ANT build script. The requirements are:

- ANT 1.8 or later
- Java Virtual Machine 1.6 later
- DocBook XSL 1.78.1 later
- Saxon 6.5.5
- Saxon 9.1.0.8

To integrate the Oxygen XML WebHelp plugin with the DocBook XSL distribution, follow these steps:

1. Download and install a `Java Virtual Machine` 1.6 or later.
2. Download and install `ANT 8.0` or later.
3. Download and unzip on your computer the DocBook XSL distribution.
   The DocBook XSL directory now contains a new subdirectory named `com.oxygenxml.webhelp` and two new files, `oxygen_custom.xsl` and `oxygen_custom_html.xsl`.
5. Download and unzip `saxon6-5-5.zip` on your computer.
6. Download and unzip `saxonb9-1-0-8j.zip` on your computer.

Registering the Oxygen XML WebHelp Plugin

To register the Oxygen XML WebHelp plugin for the DocBook XSL distribution, follow these steps:

2. In this file, copy the license key, which you purchased for your Oxygen XML WebHelp plugin.
   The WebHelp transformation process reads the Oxygen XML Developer plugin license key from this file. If the file does not exist, or it contains an invalid license, an error message is displayed.

Running a DocBook Transformation Using the WebHelp Plugin

To run a DocBook to WebHelp (`webhelp`, `webhelp-feedback`, `webhelp-mobile`) transformation using the Oxygen XML WebHelp plugin, use:

- The `docbook.bat` script file for Windows based systems.
- The `docbook.sh` script file for Unix/Linux based systems.

Note: You can call these files in an automated process or from the command line.
The docbook.bat and the docbook.sh files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate a WebHelp system, customize them to match the paths to the JVM, DocBook XSL distribution and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- **JVM_INSTALL_DIR** - specifies the path to the Java Virtual Machine installation directory on your disk.
- **ANT_INSTALL_DIR** - specifies the path to the installation directory of ANT.
- **SAXON_6_DIR** - specifies the path to the installation directory of Saxon 6.5.5.
- **SAXON_9_DIR** - specifies the path to the installation directory of Saxon 9.1.0.8.
- **DOCBOOK_XSL_DIR** - specifies the path to the installation directory of the DocBook XSL distribution.
- **TRANSTYPE** - specifies the type of the transformation you want to execute. You can set it to webhelp, webhelp-feedback and webhelp-mobile.
- **INPUT_DIR** - specifies the path to the input directory, containing the input XML file.
- **XML_INPUT_FILE** - specifies the name of the input XML file.
- **OUTPUT_DIR** - specifies the path to the output directory where the transformation output is generated.
- **DOCBOOK_XSL_DIR_URL** - specifies the path to the directory of the DocBook XSL distribution in URL format.

Additional Oxygen XML WebHelp Plugin Parameters for DocBook

You are able to append the following parameters to the command line that runs the transformation:

- **-Dwebhelp.copyright** - the copyright note (a text string value) that is added in the footer of the table of contents frame (the left side frame of the WebHelp output);
- **-Dwebhelp.footer.file** - specifies the location of a well-formed XHTML file containing your custom footer for the document body. Corresponds to the WEBHELP/footerFILE XSLT parameter. The fragment must be a well-formed XHTML, with a single root element. As a common practice, place all the content inside a <div> element;
- **-Dwebhelp.footer.include** - specifies whether the content of file set in the -Dwebhelp.footer.file is used as footer in the WebHelp pages. Its values can be yes, or no;
- **-Dwebhelp.product.id** - the value of this parameter is a text string, that the webhelp-feedback transformation requires. It represents a short name of the documentation target (product). All the user comments that are posted in the WebHelp output pages and are added in the comments database are bound to this product ID;

  **Note:** You can deploy documentation for multiple products on the same server.

- **-Dwebhelp.product.version** - the value of this parameter is a text string, that the webhelp-feedback transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.

  **Note:** Multiple documentation versions can be deployed on the same server.

In case you need to further customize your transformation, other DocBook XSL parameters can be appended. Any parameter that you want to append must follow the -D model of the above parameters. For example, you can append the html.stylesheet parameter in the following form:

- `Dhtml.stylesheet=/path/to/directory/of/stylesheet.css`

**Database Configuration for DocBook WebHelp with Feedback**

In case you run the webhelp-feedback transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the installation.html file, located at [OUTPUT_DIR]/oxygen-webhelp/resources/installation.html. The installation.html file is created by the transformation process.

**Localizing the Email Notifications of the WebHelp with Feedback System**

The WebHelp with Feedback system uses emails to notify users when comments are posted. These emails are based on templates stored in the WebHelp directory. The default messages are in English, French, German, and Japanese.
messages are copied into the WebHelp system deployment directory during the execution of the corresponding transformation scenario.

We'll suppose that you want to localize the emails into Dutch. Follow these steps:

**DocBook to WebHelp with Feedback**

- create the following directory:
  
  \[OXYGEN_DIR\]/frameworks/docbook/xsl/com.oxygenxml.webhelp/oxygen-webhelp/resources/php/templates/nl

- copy all English template files from
  
  \[OXYGEN_DIR\]/frameworks/docbook/xsl/com.oxygenxml.webhelp/oxygen-webhelp/resources/php/templates/en and paste them into the directory you just created

- edit the HTML files from the
  
  \[OXYGEN_DIR\]/frameworks/docbook/xsl/com.oxygenxml.webhelp/oxygen-webhelp/resources/php/templates/nl directory and translate the content into Dutch

- start Oxygen XML Developer plugin and edit the WebHelp with Feedback transformation scenario

  - in the Parameters tab look for the l10n.gentext.default.language parameter and set its value to the appropriate language code. In our example, use the value nl for Dutch

    Note: If you set the parameter to a value such as LanguageCode-CountryCode (for example, en-us), the transformation scenario will only use the language code

- execute the transformation scenario to obtain the WebHelp with Feedback output

**DITA to WebHelp with Feedback**

- create the following directory:
  
  \[OXYGEN_DIR\]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/php/templates/nl

- copy all English template files from
  
  \[OXYGEN_DIR\]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/php/templates/en and paste them into the directory you just created

- edit the HTML files from the
  
  \[OXYGEN_DIR\]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/php/templates/nl directory and translate the content into Dutch

- start Oxygen XML Developer plugin and edit the WebHelp with Feedback transformation scenario

  - in the Parameters tab look for the args.default.language parameter and set its value to the appropriate language code. In our example, use the value nl for Dutch

    Note: If you set the parameter to a value such as LanguageCode-CountryCode (for example, en-us), the transformation scenario will only use the language code

- execute the transformation scenario to obtain the WebHelp with Feedback output

**Adding Videos in the Output**

Videos can be included and played in all HTML5-based output formats (like WebHelp). For example, to add a YouTube video in the WebHelp output generated from DITA or DocBook documents, follow the procedures below.

**Adding Videos to WebHelp Generated from DITA Maps**

- Edit the DITA topic to reference the video using an object element like in the following example:

  ```xml
  <object outputclass="video">
    <param name="src" value="http://www.youtube.com/watch/v/VideoName"/>
  </object>
  ```

- Apply a WebHelp or WebHelp with Feedback transformation scenario to obtain the output
Adding Videos to WebHelp Generated from DocBook

- Edit the DocBook document and reference the video using a `mediaobject` element like in the following example:

```xml
<mediaobject>
  <videoobject>
    <videodata fileref="http://www.youtube.com/watch/v/VideoName"/>
  </videoobject>
</mediaobject>
```

- Apply a WebHelp or WebHelp with Feedback transformation scenario to obtain the output.

CSS Customizations

Adding your own CSS stylesheet enables you to customize the WebHelp output. To do this, edit the transformation scenario and set the `args.css` parameter to point to your custom CSS document. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.

Table of Contents Customization

The appearance of the selected item in the Table of Contents can be enhanced. To highlight the background of the selected item, go to the output folder of the WebHelp transformation and locate the `toc.css` files in the `oxygen-webhelp > resources > skins > desktop` and `oxygen-webhelp > resources > skins > desktop-frames` folders. Open them, find the `menuItemSelected` class, and change the value of the `background` property.

Note: Also, you can overwrite the same value from your own CSS.

Changing the Icons in a WebHelp Table of Contents

You can change the icons that appear in a WebHelp table of contents by assigning new image files in a custom CSS file. By default, the icons for the WebHelp table of contents are defined with the following CSS codes (the first example is the icon that appears for a collapsed menu and the second for an expanded menu):

```css
.hasSubMenuClosed{
    background: url('../img/book_closed16.png') no-repeat;
    padding-left: 16px;
    cursor: pointer;
}

.hasSubMenuOpened{
    background: url('../img/book_opened16.png') no-repeat;
    padding-left: 16px;
    cursor: pointer;
}
```

To assign different icons use the following procedure:

1. Create a custom CSS file that assigns your desired icons to the `.hasSubMenuClosed` and `.hasSubMenuOpened` properties.

```css
.hasSubMenuClosed{
    background: url('TOC-my-closed-button.png') no-repeat;
}

.hasSubMenuOpened{
    background: url('TOC-my-opened-button.png') no-repeat;
}
```

2. It is recommended that you store the image files in the same directory as the default icons: `[OXYGEN_INSTALL_DIR]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img\`.  
3. Edit the WebHelp transformation scenario and open the Parameters tab.
   a) For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes`.
   b) For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.
Adding a Logo Image in the Title Area

You are able to customize the title of your WebHelp output by using a custom CSS.

For example, to add a logo image before the title, use the following code:

```
\texttt{h1:before { \\
\quad display:inline; \\
\quad content:url('../img/myLogoImage.gif'); 
}}
```

In the example above, \texttt{myLogoImage.gif} is an image file that you place in the \texttt{[OXYGEN\_DIR]/frameworks/dita\_DITA-OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/img} directory, thus it is copied automatically by the WebHelp transformation to the output directory.

Removing the Previous/Next Links from Each WebHelp Page

The \texttt{Previous} and \texttt{Next} links that are created at the top area of each WebHelp page can be hidden with the following CSS code:

```
\texttt{.navparent, .navprev, .navnext { \\
\quad visibility:hidden; 
}}
```

Tip: Add the above code in a custom CSS stylesheet and in a WebHelp transformation scenario, set the \texttt{args.css} parameter to reference the path of this CSS stylesheet.

Adding Custom Headers and Footers

In the transformation scenario, you can use the \texttt{args.hdr} and \texttt{args.ftr} parameters to point to resources that contain your custom HTML \texttt{<div>} blocks. These are included in the header and footer of each generated topic.

To hide the horizontal separator line between the content and footer, edit the DITA transformation scenario and configure the following parameters:

- The \texttt{args.css} parameter to reference a CSS file containing the following CSS snippet:

```
\texttt{.footer_separator { \\
\quad display:none; 
}}
```

- The \texttt{args.copycss} parameter set to \texttt{true}.

Change numbering styles for ordered lists

Ordered lists (\texttt{ol}) are usually numbered in XHTML output using numerals. If you want to change the numbering to alphabetical, do the following:

1. Define a custom \texttt{outputclass} value and set it as an attribute of the ordered list, as in the following example:

```
<ol outputclass="number-alpha">
  <li>A</li>
  <li>B</li>
  <li>C</li>
</ol>
```

2. Add the following code snippet in a custom CSS file:

```
ol.number-alpha{
  list-style-type:lower-alpha;
}
```

3. Edit the DITA transformation scenario and configure the following parameters:

- \texttt{args.css} parameter to reference the custom CSS file appended earlier
- \texttt{args.copycss} parameter set to \texttt{true}.  

WebHelp Runtime Additional Parameters

A deployed WebHelp system can accept the following GET parameters:

- **log** - The value can be `true` or `false` (default value). When set to `true`, it enables JavaScript debugging.
- **contextId** - The WebHelp JavaScript engine will look up the value of this parameter in the mapping file and load the corresponding HTML help page.
- **toc.visible** - The value can be `true` (default value) or `false`. When to `false`, the table of contents will be collapsed when you load the WebHelp page.
Chapter 8

Querying Documents

Topics:

• Running XPath Expressions
• Working with XQuery

This chapter shows how to query XML documents in Oxygen XML Developer plugin with XPath expressions and the XQuery language.
Running XPath Expressions

This section covers the views, toolbars, and dialogs in Oxygen XML Developer plugin, dedicated to running XPath expressions.

What is XPath

XPath is a language for addressing specific parts of an XML document. XPath, like the Document Object Model (DOM), models an XML document as a tree of nodes. An XPath expression is a mechanism for navigating through and selecting nodes from the XML document. An XPath expression is, in a way, analogous to an SQL query used to select records from a database.

There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node.

XPath defines a library of standard functions for working with strings, numbers and boolean expressions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::*</td>
<td>Selects all children of the root node.</td>
</tr>
<tr>
<td>.//name</td>
<td>Selects all elements having the name &quot;name&quot;, descendants of the current node.</td>
</tr>
<tr>
<td>/catalog/cd[price&gt;10.80]</td>
<td>Selects all the cd elements that have a price element with a value larger than 10.80.</td>
</tr>
</tbody>
</table>

To find out more about XPath, go to [http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath).

The XPath/XQuery Builder View

The **XPath/XQuery Builder** view allows you to compose complex XPath and XQuery expressions and execute them over the currently edited XML document. For XPath 2.0 / 3.0, or XQuery expressions, you are able to use the `doc()` function to specify the source file over which the expressions are executed. When you connect to a database, the expressions are executed over that database. If you are using the **XPath/XQuery Builder** view and the current file is an XSLT document, Oxygen XML Developer plugin executes the expressions over the XML document in the associated scenario.

To open the **XPath/XQuery Builder** view, go to **Window > Show View > XPath/XQuery Builder**.

The upper part of the view contains the following actions:

- **Execute XPath** button - press this button to start the execution of the XPath or XQuery expression you are editing. The result of the execution is displayed in the **Results view** in a separate tab.

- **Favorites** button - allows you to save certain expressions that you can later reuse. To add an expression as favorite, press the star button and enter a name under which the expression is saved. The star turns yellow to confirm that the expression was saved. Expand the drop-down list next to the star button to see all your favorites. Oxygen XML Developer plugin automatically groups favorites in folders named after the method of execution.

Note: The results returned by XPath 2.0 SA and XPath 3.0 SA have a location limited to the line number of the start element (there are no column information and no end specified).

Note: Oxygen XML Developer plugin uses Saxon to execute XPath 3.0 expressions. Because Saxon implements a part of the 3.0 functions, when using a function that is not implemented, Oxygen XML Developer plugin returns a compilation error.
• **History** drop-down box - keeps a list of the last 15 executed XPath or XQuery expressions. Use the ✪ Clear history action from the bottom of the list to remove them

• **Settings** drop-down menu - contains three options:
  
  • ✪ **Update on caret move** - when enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed
  
  • ✪ **Evaluate as you type** - when you select this option, the XPath expression you are composing is evaluated in real time

  Note: The ✪ **Evaluate as you type** option and the automatic validation are disabled when the scope is other than **Current file**

• ✪ **Options** - opens the Preferences page of the currently selected processing engine

• **XPath scope** menu - Oxygen XML Developer plugin allows you to define a scope on which the XPath operation will be executed. You can choose where the XPath expression will be executed:
  
  • ✪ **Current file** - current selected file only
  
  • ✪ **Enclosing project** - all the files of the project that encloses the current edited file
  
  • ✪ **Workspace selected files** - the files selected in the workspace. The files are collected from the last selected resource provider view (Navigator, Project Explorer or Package Explorer)
  
  • ✪ **All opened files** - all files opened in the application
  
  • ✪ **Opened archive** - files open in the Archive Browser view
  
  • ✪ **Working sets** - the selected working sets

At the bottom of the scope menu there are available the following scope configuration actions:

• ✪ **Configure XPath working sets** - allows you to configure and manage collections of files and folders, encapsulated in logical containers called **working sets**

• ✪ **XPath file filter** - you can filter the files from the selected scope on which the XPath expression will be executed. By default the XPath expression will be executed only on XML files, but you can also define a set of patterns that will filter out files from the current scope.
When you hover your cursor over the XPath/XQuery version icon, a tooltip is displayed to let you know what engine Oxygen XML Developer plugin currently uses.

While you edit an XPath or XQuery expression, Oxygen XML Developer plugin assists you with the following features:

- **Content Completion Assistant** - It offers context-dependent proposals and takes into account the cursor position in the document you are editing. The set of functions proposed by the Content Completion Assistant also depends on the engine version. Select the engine version from the drop-down menu available in the toolbar.

- **Syntax highlight** - allows you to identify the components of an expression. To customize the colors of the components of the expression, open the Preferences dialog box and go to Editor > Colors.

- **Automatic validation of the expression as you type.**

  **Note:** When you type invalid syntax a red serrated line underlines the invalid fragments.

- **Function signature and documentation balloon**, when the cursor is located inside a function.

### XPath Results View

When you run an XPath expression, Oxygen XML Developer plugin displays the results of its execution in the Results View. This view contains five columns:

- **Description** - Holds the result that Oxygen XML Developer plugin displays when you run an XPath expression.
- **XPath location** - Holds the path to the matched node.
- **Resource** - Holds the name of the document on which you run the XPath expression.
- **System ID** - Holds the path to the document itself.
• **Location** - Holds the location of the result in the document.

To arrange the results depending on a column click on its header. If no information regarding location is available, Oxygen XML Developer plugin displays **Not available** in the **Location** column. Oxygen XML Developer plugin displays the results in a valid XPath expression format.

```
/node[value]/node[value]/node[value]
```

![Figure 180: XPath results highlighted in editor panel with character precision](image)

The following snippets are taken from a DocBook book based on the DocBook XML DTD. The book contains a number of chapters. To return all the chapter nodes of the book, enter `//chapter` in the XPath expression field and press **(Enter)**. This action returns all the chapter nodes of the DocBook book in the **Results View**. Click a record in the **Results View** to locate and highlight its corresponding chapter element and all its children nodes in the document you are editing.

To find all example nodes contained in the `sect2` nodes of a DocBook XML document, use the following XPath expression: `//chapter/sect1/sect2/example`. Oxygen XML Developer plugin adds a result in the **Results View** for each example node found in any `sect2` node.

For example, if the result of the above XPath expression is:

```
- /chapter[1]/sect1[3]/sect2[7]/example[1]
```

it means that in the edited file the example node is located in the first chapter, third section level one, seventh section level 2.

**Catalogs**

The evaluation of the XPath expression tries to resolve the locations of documents referenced in the expression through the **XML catalogs**. These catalogs are **configured in the Preferences** pages and **the current XInclude preferences**.

Let’s take as an example the evaluation of the `collection(URIofCollection)` function (XPath 2.0). To resolve the references from the files returned by the `collection()` function with an XML catalog, specify the class name of the XML catalog enabled parser for parsing these collection files.
The class name is `ro.sync.xml.parser.CatalogEnabledXMLReader`. Specify it as it follows:

```xml
let $docs := collection(iri-to-uri(
    "file:///D:/temp/test/XQuery-catalog/mydocsdir?recurse=yes;select=*.xml;
    parser=ro.sync.xml.parser.CatalogEnabledXMLReader")
)
```

**XPath Prefix Mapping**

To define default mappings between prefixes (that you can use in the XPath toolbar) and namespace URIs go to [XPath Options preferences panel](#) and enter the mappings in the Default prefix-name mappings table. The same preferences panel allows you to configure the default namespace used in XPath 2.0 expressions.

**Important:** If you define a default namespace, Oxygen XML Developer plugin binds this namespace to the first free prefix from the list: default, default1, default2, and so on. For example, if you define the default namespace xmlns="something" and the prefix default is not associated with another namespace, you can match tags without prefix in an XPath expression typed in the XPath toolbar by using the prefix `default`. To find all the `level` elements when you define a default namespace in the root element, execute this expression: `//default:level` in the XPath toolbar.

**Working with XQuery**

This section explains how to edit and run XQuery queries in Oxygen XML Developer plugin.

**What is XQuery**

XQuery is the query language for XML and is officially defined by a [W3C Recommendation document](#). The many benefits of XQuery include:

- XQuery allows you to work in one common model no matter what type of data you're working with: relational, XML, or object data.
- XQuery is ideal for queries that must represent results as XML, to query XML stored inside or outside the database, and to span relational and XML sources.
- XQuery allows you to create many different types of XML representations of the same data.
- XQuery allows you to query both relational sources and XML sources, and create one XML result.

**Syntax Highlight and Content Completion**

To create an XQuery document, select File > New ([Ctrl](#)+N) and when the New dialog appears select XQuery entry.

Oxygen XML Developer plugin provides syntax highlight for keywords and all known XQuery functions and operators. A content completion assistant is also available and can be activated with the ([Ctrl](#)+Space) shortcut. The functions and operators are presented together with a description of the parameters and functionality. For some supported database engines like eXist and Berkeley DB, the content completion list offers the specific XQuery functions implemented by that engine. This feature is available when the XQuery file has an associated transformation scenario which uses one of these database engines or the XQuery validation engine is set to one of them via a validation scenario or in the XQuery Preferences page.

The extension functions built in the Saxon product are available on content completion if one of the following conditions are true:

- the edited file has a transformation scenario associated that uses as transformation engine Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE
- the edited file has a validation scenario associated that use as validation engine Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE
• the validation engine specified in Preferences is Saxon 9.6.0.5 PE or Saxon 9.6.0.5 EE.

If the Saxon namespace (http://saxon.sf.net) is mapped to a prefix the functions are presented using this prefix, the default prefix for the Saxon namespace (saxon) is used otherwise.

If you want to use a function from a namespace mapped to a prefix, just type that prefix and the content completion displays all the XQuery functions from that namespace. When the default namespace is mapped to a prefix the XQuery functions from this namespace offered by content completion are also prefixed, only the function name being used otherwise.

The content completion popup window presents all the variables and functions from both the edited XQuery file and its imports.

![XQuery Content Completion](image)

**Figure 181: XQuery Content Completion**

**XQuery Outline View**

The XQuery document structure is presented in the XQuery Outline view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to a component by knowing its name. It can be opened from the Window > Show View > Other > oXygen > Outline menu action.

![XQuery Outline View](image)

**Figure 182: XQuery Outline View**

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

- **Selection update on caret move**

  Controls the synchronization between Outline view and source document. The selection in the Outline view can be synchronized with the caret moves or the changes performed in the XQuery editor. Selecting one of the components from the Outline view also selects the corresponding item in the source document.
**Sort**
Allows you to alphabetically sort the XQuery components.

**Show all components**
Displays all collected components starting from the current file. This option is set by default.

**Show only local components**
Displays the components defined in the current file only.

**Group by location/namespace/type**
Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the Outline view.

If you know the component name, you can search it in the Outline view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, (Enter), (Tab), (Shift-Tab). To switch from tree structure to the filter text field, you can use (Tab), (Shift-Tab).

**Tip:** The search filter is case insensitive. The following wildcards are accepted:
- * - any string
- ? - any character
- , - patterns separator

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

The upper part of the view contains a filter box which allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

**The XQuery Input View**
You are able to drag and drop a node in the editing area to insert XQuery expressions quickly.

![XQuery Input View](image)

**Figure 183: XQuery Input view**

**Create FLWOR by drag and drop**
For the following XML documents:

```xml
<movies>
  <movie id="1">
    <title>The Green Mile</title>
    <year>1999</year>
  </movie>
  <movie id="2">
    <title>Taxi Driver</title>
  </movie>
</movies>
```
<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>
</review>
<author>Paul</author>
</review>
<review id="101" movie-id="1">
<rating>
3
</rating>
<comment>Tom Hanks does a really nice acting.</comment>
</review>
<author>Beatrice</author>
</review>
<review id="104" movie-id="2">
<rating>
4
</rating>
<comment>Robert De Niro is my favorite actor.</comment>
</review>
<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>
{ $review/"{review/movie-id="{$movie-id}"}[@rating] text(), string($review/"{review/movie-id="{$movie-id}"}[@rating] eq 0)
|$review/"{review/movie-id="{$movie-id}"}[@rating] > 0
$movie/rating/average} }
</maxRating>
</movie>

if you drag the review element and drop it between the braces a popup menu will be displayed.

Figure 184: XQuery Input drag and drop popup menu

Select FLWOR rating and the result document will be:

XQuery Validation

With Oxygen XML Developer plugin, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.6.0.5 PE processor or the 9.6.0.5 EE, IBM DB2, eXist, Berkeley DB XML, Documentum xDb (X-Hive/DB) 10, or MarkLogic (version 5 or newer) if you installed them. Any other XQuery
processor that offers an XQJ API implementation can also be used. This is in conformance with the XQuery Working Draft. The processor is used in two cases: validation of the expression and execution. Although the execution implies a validation, it is faster to check the expression syntactically, without executing it. The errors that occurred in the document are presented in the messages view at the bottom of editor window, with a full description message. As with all error messages, if you click one entry, the line where the error appeared is highlighted.

![Figure 186: XQuery Validation](image)

Note: In case you choose a processor that does not support XQuery validation, Oxygen XML Developer plugin displays a warning when trying to validate.

When you open an XQuery document from a connection that supports validation (for example MarkLogic, or eXist), by default Oxygen XML Developer plugin uses this connection for validation. In case you open an XQuery file using a MarkLogic connection, the validation better resolves imports.

Other XQuery Editing Actions

The XQuery editor offers a reduced version of the popup menu available in the XML editor type:

- the folding actions
- the edit actions
- a part of the source actions:
  - To lower case
  - To upper case
  - Capitalize lines
- open actions:
  - Open file at Caret
  - Open file at Caret in System Application

Transforming XML Documents Using XQuery

XQueries are similar with the XSL stylesheets, both being capable of transforming an XML input into another format. You specify the input URL when you define the transformation scenario. The result can be saved and opened in the associated application. You can even run a FO processor on the output of an XQuery. The transformation scenarios may be shared between many XQuery files, are exported together with the XSLT scenarios and can be managed in the Configure Transformation Scenario dialog, or in the Scenarios view. The transformation can be performed on the XML document specified in the XML URL field, or, if this field is empty, the documents referenced from the query expression. The parameters of XQuery transforms must be set in the Parameters dialog. Parameters that are in a
namespace must be specified using the qualified name, for example a `param` parameter in the `http://www.oxygenxml.com/ns` namespace must be set with the name `{http://www.oxygenxml.com/ns}param`.

The transformation uses one of the Saxon 9.6.0.5 HE, Saxon 9.6.0.5 PE, Saxon 9.6.0.5 EE processors, a database connection (details can be found in the Working with Databases chapter - in the XQuery transformation section) or any XQuery processor that provides an XQJ API implementation.

The Saxon 9.6.0.5 EE processor supports also XQuery 3.0 transformations.

**XQJ Transformers**

This section describes the necessary procedures before running an XQJ transformation.

**How to Configure an XQJ Data Source**

Any transformer that offers an XQJ API implementation can be used when validating XQuery or transforming XML documents. An example of an XQuery engine that implements the XQJ API is Zorba.

1. In case your XQJ Implementation is native, make sure the directory containing the native libraries of the engine is added to your system environment variables: to PATH - on Windows, to LD_LIBRARY_PATH - on Linux, or to DYLD_LIBRARY_PATH - on OS X. Restart Oxygen XML Developer plugin after configuring the environment variables.

2. *Open the Preferences dialog box* and go to Data Sources.

3. Click the New button in the Data Sources panel.

4. Enter a unique name for the data source.

5. Select XQuery API for Java(XQJ) in the Type combo box.

6. Press the Add button to add XQJ API-specific files.

   You can manage the driver files using the Add, Remove, Detect, and Stop buttons.

   Oxygen XML Developer plugin detects any implementation of `javax.xml.xquery.XQDataSource` and presents it in Driver class field.

7. Select the most suited driver in the Driver class combo box.

8. Click the OK button to finish the data source configuration.

**How to Configure an XQJ Connection**

The steps for configuring an XQJ connection are the following:

1. *Open the Preferences dialog box* and go to Data Sources.

2. Click the New button in the Connections panel.

3. Enter a unique name for this connection.

4. Select one of the previously configured XQJ data sources in the Data Source combo box.

5. Fill-in the connection details.

   The properties presented in the connection details table are automatically detected depending on the selected data source.

6. Click the OK button.

**Display Result in Sequence View**

The result of an XQuery executed on a database can be very large and sometimes only a part of the full result is needed. To avoid the long time necessary for fetching the full result, select the Present as a sequence option in the Output tab of the Edit scenario dialog. This option fetches only the first chunk of the result. Clicking the More results available label that is displayed at the bottom of the Sequence view fetches the next chunk of results.

The size of a chunk can be set with the option Size limit of Sequence view. The XQuery options button from the More results available label provides a quick access to this option by opening the XQuery panel of the Preferences dialog where the option can be modified.
A chunk of the XQuery transformation result is displayed in the **Sequence** view.

**Advanced Saxon HE/PE/EE XQuery Transformation Options**

The XQuery transformation scenario allows you to configure advanced options that are specific for the Saxon HE (Home Edition), PE (Professional Edition), and EE (Enterprise Edition) engines. They are the same options as the values set in **the user preferences** but they are configured as a specific set of transformation options for each transformation scenario.

The advanced options for Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

- **Recoverable errors** ("-warnings") - Allows the user to choose how dynamic errors are handled. The following options can be selected:
  - **recover silently** ("silent") - Continues processing without reporting the error.
  - **recover with warnings** ("recover") - Issues a warning but continues processing.
• **signal the error and do not attempt recovery** ("fatal") - Issues an error and stops processing.

• **Strip whitespaces** ("-strip") - Can have one of the following values:
  - **All** ("all") - Strips all whitespace text nodes from source documents before any further processing, regardless of any xml:space attributes in the source document.
  - **Ignore** ("ignorable") - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any xml:space attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None** ("none") - Strips no whitespace before further processing.

• **Optimization level** ("-opt") - This option allows optimization to be suppressed in cases where reducing the compiling time is important, where optimization conflicts with debugging, or causes extension functions with side-effects to behave unpredictably.

• **Use linked tree model** ("-tree:linked") - This option activates the linked tree model.

• **Enable XQuery 3.0 support** ("-qversion:(1.0|3.0)") - If checked, Saxon runs the XQuery transformation with the XQuery 3.0 support (this option is enabled by default).

• **Initializer class** - Equivalent to the `-init` Saxon command-line argument. The value is the name of a user-supplied class that implements the net.sf.saxon.lib.Initializer interface. This initializer is called during the initialization process, and may be used to set any options required on the configuration programmatically. It is particularly useful for tasks such as registering extension functions, collations, or external object models, especially in Saxon-HE where the option cannot be set via a configuration file. Saxon only calls the initializer when running from the command line, but the same code may be invoked to perform initialization when running user application code.

  Important: The advanced Saxon-HE/PE/EE options configured in a transformation scenario override the Saxon-HE/PE/EE options defined globally.

The following advanced options are specific for Saxon 9.6.0.5 Professional Edition (PE) and Enterprise Edition (EE) only:

• **Use a configuration file** ("-config") - Sets a Saxon 9 configuration file that is used for XQuery transformation and validation.

• **Allow calls on extension functions** ("-ext") - If checked, calls on external functions are allowed. Checking this option is recommended in an environment where untrusted stylesheets may be executed. It also disables user-defined extension elements and the writing of multiple output files, both of which carry similar security risks.

The advanced options that are specific for Saxon 9.6.0.5 Enterprise Edition (EE) are as follows:

• **Validation of the source file** ("-val") - Requests schema-based validation of the source file and of any files read using the document() or similar functions. It can have the following values:
  - **Schema validation** ("strict") - This mode requires an XML Schema and enables parsing the source documents with strict schema-validation enabled.
  - **Lax schema validation** ("lax") - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.

• **Validation errors in the results tree treated as warnings** ("-outval") - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.

• **Generate bytecode** ("--generateByteCode:(on|off)") - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to [http://www.saxonica.com/documentation9.5/index.html#javadoc](http://www.saxonica.com/documentation9.5/index.html#javadoc).

• **Enable XQuery update** ("-update:(on|off)") - This option controls whether or not XQuery update syntax is accepted.

• **Backup files updated by XQuery** ("-backup:(on|off)") - If checked, backup versions for any XML files updated with XQuery Update are generated. This option is available when the Enable XQuery update option is enabled.
Updating XML Documents using XQuery

Using the bundled Saxon 9.6.0.5 EE XQuery processor Oxygen XML Developer plugin offers support for XQuery Update 1.0. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model. Thus, besides querying XML documents, you can modify them using the various insert/delete/modify/create methods available in the XQuery Update 1.0 standard.

Choose Saxon 9.6.0.5 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Developer plugin will notify you if the update was successful.

<table>
<thead>
<tr>
<th>Using XQuery Update to modify a tag name in an XML file</th>
</tr>
</thead>
<tbody>
<tr>
<td>rename node doc(&quot;books.xml&quot;)//publisher[1]//book[1] as &quot;firstBook&quot;</td>
</tr>
</tbody>
</table>
Chapter 9

Debugging XSLT Stylesheets and XQuery Documents

Topics:

- Overview
- Layout
- Working with the XSLT / XQuery Debugger
- Debugging Java Extensions
- Supported Processors for XSLT / XQuery Debugging

This chapter explains the user interface and how to use the debugger with XSLT and XQuery transformations.
Overview

The **XSLT Debugger** and **XQuery Debugger** perspectives enable you to test and debug XSLT 1.0 / 2.0 / 3.0 stylesheets and XQuery 1.0 / 3.0 documents including complex XPath 2.0 / 3.0 expressions. The interface presents simultaneous views of the source XML document, the XSLT/XQuery document and the result document. As you go step by step through the XSLT/XQuery document the corresponding output is generated step by step, and the corresponding position in the XML file is highlighted. At the same time, special views provide various types of debugging information and events useful to understand the transformation process.

The following set of features allow you to test and solve XSLT/XQuery problems:

- Support for XSLT 1.0 stylesheets (using Saxon 6.5.5 and Xalan XSLT engines), XSLT 2.0 / 3.0 stylesheets and XPath 2.0 / 3.0 expressions that are included in the stylesheets (using Saxon 9.6.0.5 XSLT engine) and XQuery 1.0 / 3.0 (using Saxon 9.6.0.5 XQuery engine).
- Stepping capabilities: step in, step over, step out, run, run to cursor, run to end, pause, stop.
- Output to source mapping between every line of output and the instruction element / source context that generated it.
- Breakpoints on both source and XSLT/XQuery documents.
- Call stack on both source and XSLT/XQuery documents.
- Trace history on both source and XSLT/XQuery documents.
- Support for XPath expression evaluation during debugging.
- Step into imported/included stylesheets as well as included source entities.
- Available templates and hits count.
- Variables view.
- Dynamic output generation.

Layout

The XML and XSL files are displayed in **Text mode**. The **Grid mode** is available only in the **Editor perspective**.

The debugger perspective contains four sections:

- **Source document view (XML)** - Displays and allows the editing of XML files (documents).
- **XSLT/XQuery document view (XSLT/XQuery)** - Displays and allows the editing of XSL files (stylesheets) or XQuery documents.
- **Output document view** - Displays the output that results from inputting a document (XML) and a stylesheet (XSL) or XQuery document in the transformer. The transformation result is written dynamically while the transformation is processed.
- **Control view** - The control view is used to configure and control the debugging operations. It also provides a set of **Information views** types. This pane has two sections:
  - **Control toolbar**
  - **Information views**
XML documents and XSL stylesheets or XQuery documents that were opened in the Editor perspective are automatically sorted into the first two panes. When multiple files of each type are opened, the individual documents and stylesheets are separated using the familiar tab management system of the Editor perspective. Selecting a tab brings the document or stylesheet into focus and enables editing without the need to go back to the Editor perspective.

During debugging, the current execution node is highlighted in both document (XML) and XSLT/XQuery views.

**Control Toolbar**

The **Control** toolbar contains all the actions that you need to configure and control the debugging process. The following actions are described as they appear in the toolbar from left to right.

**Figure 190: Control Toolbar**

**XML source selector**

The current selection represents the source document used as input by the transformation engine. A drop-down list contains all opened files (XML files being emphasized). This option allows you to use other file types also as source documents. In an XQuery debugging session this selection field can be set to the default value NONE, because usually XQuery documents do not require an input source.

**XSL / XQuery selector**

The current selection represents the stylesheet or XQuery document to be used by the transformation engine. The selection list contains all opened files (XSLT / XQuery files being emphasized).
**Link with editor**
When enabled, the XML and XSLT/XQuery selectors display the names of the files opened in the central editor panels. This button is disabled by default.

**Output selector**
The selection represents the output file specified in the associated transformation scenario.

**XSLT / XQuery parameters**
XSLT / XQuery parameters to be used by the transformation.

**Libraries**
Add and remove the Java classes and jars used as XSLT extensions.

**Turn on profiling**
Enables / Disables current transformation profiling.

**Enable XHTML output**
Enables the rendering of the output in the XHTML output view during the transformation process. For performance issues, disable XHTML output when working with very large files. Note that only XHTML conformant documents can be rendered by this view. In order to view the output result of other formats, such as HTML, save the Text output area to a file and use an external browser for viewing.

When starting a debug session from the editor perspective using the Debug Scenario action, the state of this toolbar button reflects the state of the Show as XHTML output option from the scenario.

**Turn on/off output to source mapping**
Enables or disables the output to source mapping between every line of output and the instruction element / source context that generated it.

**Debugger preferences**
Quick link to Debugger preferences page.

**XSLT / XQuery engine selector**
Lists the processors available for debugging XSLT and XQuery transformations.

**XSLT / XQuery engine advanced options**
Advanced options available for Saxon 9.6.0.5.

**Step into F7**
Starts the debugging process and runs until the next instruction is encountered.

**Step over F8 (Alt F7 on OS X)**
Run until the current instruction and its sub-instructions are over. Usually this will advance to the next sibling instruction.

**Step out Shift F7**
Run until the parent of the current instruction is over. Usually this will advance to the next sibling of the parent instruction.

**Run**
Starts the debugging process. The execution of the process is paused when a breakpoint is encountered or the transformation ends.

**Run to cursor Ctrl F5 (Command F5 on OS X)**
Starts the debugging process and runs until one of the following conditions occur: the line of cursor is reached, a valid breakpoint is reached or the execution ends.
Run to end Alt F5
Runs the transformation until the end, without taking into account enabled breakpoints, if any.

Pause Shift F6
Request to pause the current transformation as soon as possible.

Stop F6
Request to stop the current transformation without completing its execution.

Show current execution nodes
Reveals the current debugger context showing both the current instruction and the current node in the XML source. Possible displayed states:

• entering (↑) or leaving (↓) an XML execution node
• entering (↑) or leaving (↓) an XSL execution node
• entering (↑) or leaving (↓) an XPath execution node

Note: When you set a MarkLogic server as a processor, the Show current execution nodes button is named Refresh current session context from server. Click this button to refresh the information in all the views.

Note: For some of the XSLT processors (Saxon-HE/PE/EE) the debugger could be configured to step into the XPath expressions affecting the behavior of the following debugger actions: Step into, Step over or Step Out.

Information View

The Information view is comprised of two panes that are used to display various types of information used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This enables the developer to obtain a clear view of the transformation progress. Using the debug controls developers can easily isolate parts of stylesheet therefore they may be understood and modified. The information types include:

Left side information views

• Context Node view
• XWatch view
• Breakpoints view
• Messages view (XSLT only)
• Variables view

Right side information views

• Stack view
• Trace view
• Templates view (XSLT only)
• Nodeset view

Context Node View

The context node is valid only for XSLT debugging sessions and is a source node corresponding to the XSL expression that is evaluated. It is also called the context of execution. The context node implicitly changes as the processor hits various steps (at the point where XPath expressions are evaluated). This node has the same value as evaluating '.' (dot) XPath expression in XWatch view. The value of the context node is presented as a tree in the view.
The context node is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix, the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel. The Context view also presents the current mode of the XSLT processor in case this mode differs from the default one.

**XPath Watch (XWatch) View**

The XWatch view shows XPath expressions evaluated during the debugging process. Expressions are evaluated dynamically as the processor changes its source context.

When you type an XPath expression in the Expression column, Oxygen XML Developer plugin supports you with syntax highlight and content completion assistance.

**Table 4: XWatch columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>XPath expression to be evaluated (XPath 1.0 or 2.0 / 3.0 compliant).</td>
</tr>
<tr>
<td>Value</td>
<td>Result of XPath expression evaluation. Value has a type (see the possible values in the section Variables View on page 392). For Node Set results, the number of nodes in the set is shown in parenthesis.</td>
</tr>
</tbody>
</table>

**Important:** Remarks about working with the XWatch view:

- Expressions that reference variable names are not evaluated.
- The expression list is not deleted at the end of the transformation (it is preserved between debugging sessions).
- To insert a new expression, click the first empty line of the Expression column and start typing.
To delete an expression, click on its **Expression** column and delete its content.

If the expression result type is a **Node Set**, click it (**Value** column) and its value is displayed in the **Nodes/Values Set view**.

---

**Breakpoints View**

This view lists all breakpoints that are set on opened documents. Once you **insert a breakpoint** it is automatically added in this list. Breakpoints can be set in XSLT/XQuery documents and XML documents in XSLT/XQuery debugging sessions. A breakpoint can have an associated break condition that represents an XPath expression evaluated in the current debugger context. In order to be processed, their evaluation result should be a boolean value. A breakpoint with an associated condition only stops the execution of the Debugger if the breakpoint condition is evaluated as **true**.

![Breakpoints View](image)

**Figure 193: The Breakpoints View**

The **Breakpoints** view contains the following columns:

- **Enabled** - If checked, the current condition is evaluated and taken into account.
- **Resource** - Resource file and number of the line where the breakpoint is set.
- **Condition** - XSLT/XQuery expression to be evaluated during debugging. The expression will be evaluated at every debug step.

Clicking a record highlights the breakpoint line in the document.

*Note:* The breakpoints list is not deleted at the end of a transformation (it is preserved between debugging sessions).

The following actions are available in the contextual menu of the table:

- **Go to**
  - Moves the cursor to the source of the breakpoint.
- **Enable**
  - Enables the breakpoint.
- **Disable**
  - Disables the breakpoint. A disabled breakpoint will not be evaluated by the Debugger.
- **Add**
  - Allows you to add a new breakpoint and breakpoint condition.
- **Edit**
  - Allows you to edit an existing breakpoint.
- **Remove**
  - Deletes the selected breakpoint.
- **Enable all**
  - Enables all breakpoints.
- **Disable all**
  - Disables all breakpoints.
Remove all

Removes all breakpoints.

**Messages View**

`xsl:message` instructions are one way to signal special situations encountered during transformation as well as a raw way of doing the debugging. This view is available only for XSLT debugging sessions and shows all `xsl:message` calls executed by the XSLT processor during transformation.

![Figure 194: The Messages View](image)

<table>
<thead>
<tr>
<th>Message</th>
<th>Terminate</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message 1</td>
<td>no</td>
<td><code>personal.xsl [line:8]</code></td>
</tr>
<tr>
<td>Message 2</td>
<td>no</td>
<td><code>personal.xsl [line:12]</code></td>
</tr>
<tr>
<td>Message 3</td>
<td>no</td>
<td><code>personal.xsl [line:29]</code></td>
</tr>
</tbody>
</table>

**Table 5: Messages columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>Message content.</td>
</tr>
<tr>
<td>Terminate</td>
<td>Signals if processor terminates the transformation or not once it encounters the message (yes/no respectively).</td>
</tr>
<tr>
<td>Resource</td>
<td>Resource file where <code>xsl:message</code> instruction is defined and the message line number.</td>
</tr>
</tbody>
</table>

The following actions are available in the contextual menu:

**Go to**

Highlight the XSL fragment that generated the message.

**Copy**

Copies to clipboard message details (system ID, severity info, description, start location, terminate state).

**Important:** Remarks

- Clicking a record from the table highlights the `xsl:message` declaration line.
- Message table values can be sorted by clicking the corresponding column header. Clicking the column header switches the sorting order between: ascending, descending, no sort.

**Stack View**

This view shows the current execution stack of both source and XSLT/XQuery nodes. During transformation two stacks are managed: one of source nodes being processed and the other for XSLT/XQuery nodes being processed. Oxygen XML Developer plugin shows both node types into one common stack. The source (XML) nodes are preceded by a red color icon while XSLT/XQuery nodes are preceded by a green color icon. The advantage of this approach is that you can always see the source scope on which a XSLT/XQuery instruction is executed (the last red color node on the stack). The stack is oriented upside down.
Figure 195: 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

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Order number, represents the depth of the node (0 is the stack base).</td>
</tr>
<tr>
<td>XML/XSLT/XQuery Node</td>
<td>Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as #document.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Attributes of the node (a list of id=&quot;value&quot; pairs).</td>
</tr>
<tr>
<td>Resource</td>
<td>Resource file where the node is located.</td>
</tr>
</tbody>
</table>

**Important:** Remarks:
- Clicking a record from the stack highlights that node’s location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: xsl:template using Saxon and template using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

Output Mapping Stack View

The Output Mapping Stack view displays context data and presents the XSLT templates/XQuery elements that generated specific areas of the output.

Figure 196: The Output Mapping Stack view
The Go to action of the contextual menu takes you in the editor panel at the line containing the XSLT element displayed in the **Output Mapping Stack** view.

### Table 7: Output Mapping Stack columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>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.</td>
</tr>
<tr>
<td>XSL/XQuery Node</td>
<td>The name of an XSLT template/XQuery element that participated in the generation of the selected output area.</td>
</tr>
<tr>
<td>Attributes</td>
<td>The attributes of the XSLT template/XQuery node.</td>
</tr>
<tr>
<td>Resource</td>
<td>The name of the file containing the XSLT template/XQuery element.</td>
</tr>
</tbody>
</table>

**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 197: The Trace History View](image)

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**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>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.</td>
</tr>
<tr>
<td>XML/XSLT/XQuery Node</td>
<td>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).</td>
</tr>
<tr>
<td>Attributes</td>
<td>Attributes of the node (a list of id=&quot;value&quot; pairs).</td>
</tr>
<tr>
<td>Resource</td>
<td>Resource file where the node is located.</td>
</tr>
</tbody>
</table>

**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 198: 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**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>The <code>match</code> attribute of the <code>xsl:template</code>.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hits</td>
<td>The number of hits for the <code>xsl:template</code>. Shows how many times the XSLT processor used this particular template.</td>
</tr>
<tr>
<td>Priority</td>
<td>The template priority as established by XSLT processor.</td>
</tr>
<tr>
<td>Mode</td>
<td>The <code>mode</code> attribute of the <code>xsl:template</code>.</td>
</tr>
<tr>
<td>Name</td>
<td>The <code>name</code> attribute of the <code>xsl:template</code>.</td>
</tr>
<tr>
<td>Resource</td>
<td>The resource file where the template is located.</td>
</tr>
</tbody>
</table>

**Important:** Remarks:
- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

**Nodes/Values Set View**

This view is always used in relation with the Variables view and the XWatch view. It shows an XSLT node set value in a tree form. The node set view is updated as response to the following events:

- You click a variable having a node set value in one of the above 2 views.
- You click a tree fragment in one of the above 2 views.
- You click an XPath expression evaluated to a node set in one of the above 2 views.

**Important:** Remarks:
- In case of longer values in the right side panel the interface shows three suspension points (…) at the end. A more detailed value is available as tooltip.
- Clicking a record highlights the location of that node into the source or stylesheet view.

**Variables View**

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Developer plugin uses the following icons to differentiate variables and parameters:

- `V` - Global variable.
- `{V}` - Local variable.
The following value types are available:

- **Boolean**
- **String**
- **Date** - XSLT 2.0 / 3.0 only.
- **Number**
- **Set**
- **Object**
- **Fragment** - Tree fragment.
- **Any**
- **Undefined** - The value was not yet set, or it is not accessible.

**Note:**

When Saxon 6.5 is used, if the value is unavailable, then the following message is displayed in the **Value** field: "The variable value is unavailable".

When Saxon 9 is used:

- If the variable is not used, the **Value** field displays "The variable is declared but never used".
- If the variable value cannot be evaluated, the **Value** field displays "The variable value is unavailable".

- **Document**
- **Element**
- **Attribute**
- **ProcessingInstruction**
- **Comment**
- **Text**
- **Namespace**
- **Evaluating** - Value under evaluation.
- **Not Known** - Unknown types.

**Figure 200:** The Variables View

**Table 10:** Variables columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of variable / parameter.</td>
</tr>
<tr>
<td>Value type</td>
<td>Type of variable/parameter.</td>
</tr>
<tr>
<td>Value</td>
<td>Current value of variable / parameter.</td>
</tr>
</tbody>
</table>
The value of a variable (the Value column) can be copied to the clipboard for pasting it to other editor area with the action Copy value from the contextual menu of the table from the view. This is useful in case of long and complex values which are not easy to remember by looking at them once.

Important: Remarks:

• Local variables and parameters are the first entries presented in the table.
• Clicking a record highlights the variable definition line.
• Variable values could differ depending on the transformation engine used or stylesheet version set.
• If the value of the variable is a node set or a tree fragment, clicking on it causes the Node Set view to be shown with the corresponding set of values.
• Variable table values can be sorted by clicking the corresponding column header. Clicking the column header switches between the orders: ascending, descending, no sort.

Multiple Output Documents in XSLT 2.0 and XSLT 3.0

For XSLT 2.0 and XSLT 3.0 stylesheets that store the output in more than one file by using the xsl:result-document instruction the content of the file created in this way is displayed dynamically while the transformation is running in an output view. There is one view for each xsl:result-document instruction so that the output of different instructions is not mixed but is presented in different views.

Working with the XSLT / XQuery Debugger

This section describes how to work with the debugger in the most common use cases.

To watch our video demonstration about how you can use the XSLT Debugger, go to http://oxygenxml.com/demo/XSLT_Debugger.html.

Steps in a Typical Debug Process

To debug a stylesheet or XQuery document follow the procedure:

1. Open the source XML document and the XSLT/XQuery document.
2. If you are in the Oxygen XML Developer plugin XML perspective switch to the Oxygen XML Developer plugin XSLT Debugger perspective or the Oxygen XML Developer plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):
   • Menu Window > Open Perspective > Other ... > Oxygen XSLT Debugger
   • The toolbar button Debug scenario - This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.
3. Select the source XML document in the XML source selector of the Control toolbar. In case of XQuery debugging if your XQuery document has no implicit source set the source selector value to NONE.
4. Select the XSLT/XQuery document in the XSLT/XQuery selector of the Control toolbar.
5. Set XSLT/XQuery parameters from the button available on the Control toolbar.
6. Set one or more breakpoints.
7. Step through the stylesheet using the buttons available on the Control toolbar:
   • Step into
   • Step over
   • Step out
   • Run
   • Run to cursor
8. Examine the information in the Information views to find the bug in the transformation process.

You may find the procedure for determining the XSLT template/XQuery element that generated an output section useful for fixing bugs in the transformation.

Using Breakpoints

The Oxygen XML Developer plugin XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure breakpoints are persistent between work sessions, they are saved at project level. You can set a maximum of 100 breakpoints per project.

Inserting Breakpoints

To insert a breakpoint, follow these steps:

1. Click the line where you want to insert the breakpoint in the XML source document or the XSLT/XQuery document. You can only set breakpoints on the XML source in XSLT or XQuery debugging sessions.

Breakpoints are automatically created on the ending line of a start tag, even if you click a different line.

2. Right-click on the vertical stripe on the left side of the editor panel and select Add breakpoint.

Removing Breakpoints

Only one action is required to remove a breakpoint:

Right-click the breakpoint icon in the vertical stripe on the left side of the editor panel and select Remove breakpoint.

Determining What XSLT / XQuery Expression Generated Particular Output

In order to quickly spot the XSLT templates or XQuery expressions with problems it is important to know what XSLT template in the XSLT stylesheet or XQuery expression in the XQuery document and what element in the source XML document generated a specified area in the output.

Some of the debugging capabilities, for example Step in can be used for this purpose. Using Step in you can see how output is generated and link it with the XSLT/XQuery element being executed in the current source context. However, this can become difficult on complex XSLT stylesheets or XQuery documents that generate a large output.

You can click on the text from the Text output view or XHTML output view and the editor will select the XML source context and the XSLT template/XQuery element that determined the state of the XSLT/XQuery processor at the moment of generating the specified output area speeds up the debugging process.

1. Switch to the Oxygen XML Developer plugin XSLT Debugger perspective or Oxygen XML Developer plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):

   - Go to menu Window > Open Perspective > Other ... > Oxygen XSLT Debugger
   - Go to menu . The toolbar button . This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer
2. Select the source XML document in the XML source selector of the Control toolbar. In case of XQuery debugging without an implicit source choose the NONE value.

3. Select the XSLT / XQuery document in the XSLT / XQuery selector of the Control toolbar.

4. Select the XSLT / XQuery engine in the XSLT / XQuery engine selector of the Control toolbar.

5. Set XSLT / XQuery parameters from the button available on the Control toolbar.

6. Apply the XSLT stylesheet or XQuery transformation using the button Run to end available on the Control toolbar.

7. Inspect the mapping by clicking a section of the output from the Output view of the Oxygen XML Developer plugin XSLT Debugger or Oxygen XML Developer plugin XQuery Debugger perspectives.

**Figure 201: Text Output to Source Mapping**

This action will highlight the XSLT / XQuery element and the XML source context. This XSLT template/XQuery element that is highlighted in the XSLT/XQuery editor represents only the top of the stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the clicked output section. In case of complex transformations inspecting the whole stack of XSLT templates/XQuery elements speeds up the debugging process. This stack is available in the Output Mapping Stack view.
Debugging Java Extensions

The XSLT/XQuery debugger does not step into Java classes that are configured as XSLT/XQuery extensions of the transformation. To step into Java classes, inspect variable values and set breakpoints in Java methods, set up a Java debug configuration in an IDE like the Eclipse SDK as described in the following steps:

1. Create a debug configuration.
   a) Set at least 256 MB as heap memory for the Java virtual machine (recommended 512 MB) by setting the `-Xmx` parameter in the debug configuration, for example “-Xmx512m”.
   b) Make sure the `[OXYGEN_DIR]/lib/oxygen.jar` file and your Java extension classes are on the Java classpath.
      The Java extension classes should be the same classes that were set as an extension of the XSLT/XQuery transformation in the debugging perspective.
   c) Set the class `ro.sync.exml.Oxygen` as the main Java class of the configuration.
      The main Java class `ro.sync.exml.Oxygen` is located in the `oxygen.jar` file.

2. Start the debug configuration.
   Now you can set breakpoints and inspect Java variables as in any Java debugging process executed in the selected IDE (Eclipse SDK, and so on.).

Supported Processors for XSLT / XQuery Debugging

The following built-in XSLT processors are integrated in the debugger and can be selected in the Control Toolbar:

- Saxon 9.6.0.5 HE (Home Edition) - a limited version of the Saxon 9 processor, capable of running XSLT 1.0, XSLT 2.0 / 3.0 basic and XQuery 1.0 transformations, available in both the XSLT debugger and the XQuery one,
- Saxon 9.6.0.5 PE (Professional Edition) - capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery one,
- Saxon 9.6.0.5 EE (Enterprise Edition) - a schema aware processor, capable of running XSLT 1.0 transformations, XSLT 2.0 / 3.0 basic ones, XSLT 2.0 / 3.0 schema aware ones and XQuery 1.0 / 3.0 ones, available in both the XSLT debugger and the XQuery debugger,
- Saxon 6.5.5 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger,
- Xalan 2.7.1 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger.
Chapter 10

Performance Profiling of XSLT Stylesheets and XQuery Documents

Topics:
- Overview
- Viewing Profiling Information
- Working with XSLT/XQuery Profiler

This chapter explains the user interface and how to use the profiler for finding performance problems in XSLT transformations and XQuery ones.
Overview

Whether you are trying to identify a performance issue that is causing your production XSLT/XQuery transformation to not meet customer expectations or you are trying to proactively identify issues prior to deploying your XSLT/XQuery transformation, using the XSLT/XQuery profiler feature is essential to helping you save time and ultimately ensure a better performing, more scalable XSLT/XQuery transformation.

The XSLT/XQuery profiling feature can use any available XSLT/XQuery processors that could be used for debugging and it is available from the debugging perspective.

Enabling and disabling the profiler is controlled by the “Profiler button from the debugger control toolbar. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you should set it before starting the transformation.

Viewing Profiling Information

This section explains the views that display the profiling data collected by the profiles during the transformation.

Invocation Tree View

This view shows a top-down call tree representing how XSLT instructions or XQuery expressions are processed.

Figure 202: Invocation tree view

The entries in the invocation tree have different meanings which are indicated by the displayed icons:

- Points to a call whose inherent time is insignificant compared to its total time.
- Points to a call whose inherent time is significant compared to its total time (greater than 1/3rd of its total time).

Every entry in the invocation tree has textual information attached which depends on the XSLT/XQuery profiler settings:

- A percentage number of total time which is calculated with respect to either the root of the tree or the calling instruction.
- A total time measurement in milliseconds or microseconds. This is the total execution time that includes calls into other instructions.
- A percentage number of inherent time which is calculated with respect to either the root of the tree or the calling instruction.
- An inherent time measurement in milliseconds or microseconds. This is the inherent execution time of the instruction.
- An invocation count which shows how often the instruction has been invoked on this call-path.
- An instruction name which contains also the attributes description.

Hotspots View

This view shows a list of all instruction calls which lie above the threshold defined in the XSLT/XQuery profiler settings.
By opening a hotspot instruction entry, the tree of back-traces leading to that instruction call are calculated and shown. Every hotspot is described by the values from the following columns:

- The instruction name.
- The inherent time in milliseconds or microseconds of how much time has been spent in the hotspot together with a bar whose length is proportional to this value. All calls into this instruction are summed up regardless of the particular call sequence.
- The invocation count of the hotspot.

If you click on the handle on the left side of a hotspot, a tree of back-traces will be shown. Every entry in the backtrace tree has textual information attached to it which depends on the XSLT/XQuery profiler settings:

- A percentage number which is calculated with respect either to the total time or the called instruction.
- A time measured in milliseconds or microseconds of how much time has been contributed to the parent hotspot on this call-path.
- An invocation count which shows how often the hotspot has been invoked on this call-path.

Note: This is not the number of invocations of this instruction.

- An instruction name which contains also its attributes.

### Working with XSLT/XQuery Profiler

Profiling activity is linked with debugging activity, so the first step in order to profile is to switch to debugging perspective and follow the corresponding procedure for debugging (see Working with XSLT Debugger).

Immediately after turning the profiler on two new information views are added to the current debugger information views:

- **Invocation tree view** on left side
- **Hotspots view** on right side

Profiling data is available only after the transformation ends successfully.

Looking to the right side (Hotspots view), you can immediately spot the time the processor spent in each instruction. As an instruction usually calls other instructions the used time of the called instruction is extracted from the duration time of the caller (the hotspot only presents the inherent time of the instruction).

Looking to the left side (Invocation tree view), you can examine how style instructions are processed. This result view is also named call-tree, as it represents the order of style processing. The profiling result shows the duration time for each of the style-instruction including the time needed for its called children.
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 causes Oxygen XML Developer plugin to highlight the XSLT stylesheet or XQuery expression source line where the instruction is defined.

When navigating through the trees by opening instruction calls, Oxygen XML Developer plugin automatically expands instructions which are only called by one other instruction themselves.

The profiling data can be saved into XML and HTML format. On any view you should right-click, use the pop-up menu and select the corresponding choice. Basically saving HTML means saving XML and applying an XSLT stylesheet to render the report as XML. These stylesheets are included in the Oxygen XML Developer plugin distribution (see the subfolder [OXYGEN_DIR]/frameworks/profiler/) so you can make your own report based on the profiling raw data.

If you like to change the XSLT/XQuery profiler settings you should right-click on view, use the pop-up menu and choose the corresponding View settings entry.

Caution: Profiling exhaustive transformation may run into an OutOfMemory error due to the large amount of information being collected. If this is the case you can close unused projects when running the profiling or use high values for Java VM options -Xms and -Xmx. If this does not help you can shorten your source XML file and try again.

To watch our video demonstration about the XSLT/XQuery Profiler, go to http://oxygenxml.com/demo/XSLT_Profiling.html.
Chapter 11

Working with Archives

Topics:
- Browsing and Modifying Archive Structure
- Working with EPUB
- Editing Files From Archives

Oxygen XML Developer plugin offers the means to manipulate files directly from ZIP type archives. By manipulation one should understand opening and saving files directly in archives, browsing and modifying archive structures. The archive support is available for all ZIP-type archives, which includes:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

This means that you can modify, transform, validate files directly from OOXML or ODF packages. The structure and content of an EPUB book, OOXML file or ODF file can be opened, edited and saved as for any other ZIP archive.

You can transform, validate and perform many other operations on files directly from an archive. When selecting an URL for a specific operation like transformation or validation you can click the Browse for archived file button to navigate and choose the file from a certain archive.
Browsing and Modifying Archive Structure

You can navigate archives in the Archives Browser either by opening them from the Navigator or by using the integration with the Eclipse File System. For the EFS (Eclipse File System) integration you must right click the archive in the Navigator and choose Expand Zip Archive. All the standard Eclipse Navigator actions are available on the mounted archive. If you decide to close the archive you can use the Collapse ZIP Archive action located in the contextual menu for the expanded archive. Any file opened from the archive expanded in the EFS will be closed when the archive in unmounted.

If you open an archive as an Eclipse editor, the archive will be unmounted when the editor is closed.

⚠️ **Important:** If a file is not recognized by Oxygen XML Developer plugin as a supported archive type, you can add it from the Archive preferences page.

The following operations are available on the Archive Browser toolbar:

- **Reopen**
  You can use this drop-down to reopen recently edited archives. Apart from the history of the recently edited archives, the drop-down also contains the Clear history and Open Archive actions.

- **New folder...**
  Creates a folder as child of the selected folder in the browsed archive.

- **New file...**
  Creates a file as child of the selected folder in the browsed archive.

- **Add files...**
  Adds existing files as children of the selected folder in the browsed archive.

  Note: You can also add files in the archive by dragging them from the file browser or Project view and dropping them in the Archive Browser view.

- **Delete**
  Deletes the selected resource in the browsed archive.

- **Archive Options...**
  Opens the Archive preferences page.

The following additional operations are available from the Archive Browser contextual menu:

- **Open**
  Opens a resource from the archive in the editor.

- **New folder...**
  Creates a folder as child of the selected folder in the browsed archive.

- **New file...**
  Creates a file as child of the selected folder in the browsed archive.

- **Add files...**
  Adds existing files as children of the selected folder in the browsed archive.

  Note: On OS X, there is also available the Add file... action, which allows you to add one file at a time.

- **Find/Replace in Files**
  Allows you to search for and replace specific pieces of text inside the archive.
Cut
Cut the selected archive resource.

Copy
Copy the selected archive resource.

Paste
Paste a file or folder into the archive.

Note: You can add files in the archive by copying the files from the Project view and paste them into the Archive view.

Delete
Remove a file or folder from archive.

Copy location
Copies the URL location of the selected resource.

Refresh
Refreshes the selected resource.

Properties
Views properties for the selected resource.

Working with EPUB

EPUB is a free and open electronic book standard by the International Digital Publishing Forum (IDPF). It was designed for reflowable content, meaning that the text display can be optimized for the particular display device used by the reader of the EPUB-formatted book. Oxygen XML Developer plugin supports both EPUB 2.0 and EPUB 3.0.

Opening an EPUB file exposes all its internal components:

- Document content (XHTML and image files).
- Packaging files.
- Container files.
Figure 205: EPUB file displayed in Eclipse

Here you can edit, delete and add files that compose the EPUB structure. To check that the EPUB file you are editing is valid, invoke the Validate and Check for Completeness action. Oxygen XML Developer plugin uses the open-source EpubCheck validator to perform the validation. This validator detects many types of errors, including OCF container structure, OPF and OPS mark-up, as well as internal reference consistency. All errors found during validation are displayed in a separate tab in the Errors view.

**Note:** Invoke the Open in System Application action to see how the EPUB is rendered in your system default EPUB reader application.

**Note:** All changes made to the structure of an EPUB, or to the contents of the files inside an EPUB are immediately saved.

To watch our video demonstration about the EPUB support in Oxygen XML Developer plugin, go to http://oxygenxml.com/demo/Epub.html.

Create an EPUB

To begin writing an EPUB file from scratch, do the following:

1. Go to File > New, press Ctrl N (Command N on OS X) on your keyboard. or click New on the main toolbar.
2. Choose EPUB Book template. Click Create. Choose the name and location of the file. Click Save. A skeleton EPUB file is saved on disk and open in the Archive Browser view.
3. Use the Archive Browser view specific actions to edit, add and remove resources from the archive.
4. Use the Validate and Check for Completeness action to verify the integrity of the EPUB archive.
### Publish to EPUB

Oxygen XML Developer plugin comes with built-in support for publishing DocBook and DITA XML documents directly to EPUB.

1. Open the **Configure Transformation Scenario(s)** dialog box and select a predefined transformation scenario. To publish from DITA, select the **DITA Map EPUB** transformation scenario. To publish from DocBook select the **DocBook EPUB** transformation scenario.

2. Click **Apply associated** to run the transformation scenario.

### Editing Files From Archives

You can open and edit files directly from an archive using the **Archive Browser** view. When saving the file back to archive, you are prompted to choose if you want the application to make a backup copy of the archive before saving the new content. If you choose **Never ask me again**, you will not be asked again to make backup copies. You can re-enable the pop-up message from the **Archive preferences page**.

**Note:** All changes made to the structure of an archive, or to the contents of the files inside an archive are immediately saved.
Chapter 12

Working with Databases

Topics:
- Relational Database Support
- Native XML Database (NXD) Support
- XQuery and Databases
- WebDAV Connection
- BaseX Support

XML is a storage and interchange format for structured data and is supported by all major database systems. Oxygen XML Developer plugin offers the means for managing the interaction with some of the most commonly used databases (both relational and Native XML Databases). Through this interaction, Oxygen XML Developer plugin helps users to understand browsing, querying, SQL execution support, content editing, importing from databases, and generating XML Schema from database structure.
Relational Database Support

Relational databases use a relational model and are based on tables linked by a common key. Oxygen XML Developer plugin offers support for the following relational databases: IBM DB2, MySQL, Microsoft SQL Server, and Oracle 11g.

The following actions are allowed:

- Browsing the tables of these types of databases in the Data Source Explorer view
- Executing SQL queries against them
- Calling stored procedures with input and output parameters

Oxygen XML Developer plugin offers generic support (table browsing and execution of SQL queries) for any JDBC-compliant database (for example, MariaDB).

To watch our video demonstration about the integration between the relational databases and Oxygen XML Developer plugin, go to http://www.oxygenxml.com/demo/Author_Database_Integration.html.

Configuring Database Data Sources

This section describes the procedures for configuring the data sources for relational databases:

- IBM DB2
- Microsoft SQL Server
- Generic JDBC
- MySQL
- Oracle 11g
- PostgreSQL 8.3

Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

- IBM DB2
- Microsoft SQL Server
- JDBC-ODBC
- MySQL
- Generic ODBC
- Oracle 11g
- PostgreSQL 8.3

How to Configure Support For Relational Databases

This section contains procedures about configuring the support for various relational databases.

How to Configure IBM DB2 Support

To configure the support for the IBM DB2 database follow this procedure:

1. Go to the IBM website and in the DB2 Clients and Development Tools category select the DB2 Driver for JDBC and SQLJ download link. Fill out the download form and download the zip file. Unzip the zip file and use the db2jcc.jar and db2jcc_license_cu.jar files in Oxygen XML Developer plugin for configuring a DB2 data source.
2. Configure a IBM DB2 Data Source driver.
3. Configure a IBM DB2 Server Connection.
4. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).
How to Configure an IBM DB2 Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an IBM DB2 server are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Data Sources panel.

The dialog box for configuring a data source is opened.

![Figure 206: Data Source Drivers Configuration Dialog Box](image)

3. Enter a unique name for the data source.
4. Select DB2 in the driver Type drop-down list.
5. Add the driver files for IBM DB2 using the Add Files button.

   The IBM DB2 driver files are:
   - db2jcc.jar
   - db2jcc_license_cisuz.jar
   - db2jcc_license_cu.jar

   The Driver files section lists download links for database drivers that are necessary for accessing IBM DB2 databases in Oxygen XML Developer plugin.

6. Select the most appropriate Driver class.
7. Click the OK button to finish the data source configuration.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to http://www.oxygenxml.com/demo/DB2.html.

How to Configure an IBM DB2 Connection

The support to create an IBM DB2 connection is available in the Enterprise edition only.
To configure a connection to an IBM DB2 server, follow these steps:

1. *Open the Preferences dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

   ![The Connection Configuration Dialog Box](image)

3. Enter a unique name for the connection.
4. Select an *IBM DB2* data source in the **Data Source** drop-down list.
5. Enter the connection details.
   a) Enter the URL to the installed IBM DB2 engine.
   b) Enter the user name to access the IBM DB2 engine.
   c) Enter the password to access the IBM DB2 engine.
6. Click the **OK** button to finish the configuration of the database connection.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to [http://www.oxygenxml.com/demo/DB2.html](http://www.oxygenxml.com/demo/DB2.html).

### How to Configure Microsoft SQL Server Support

To configure the support for Microsoft SQL Server database follow this procedure:

2. *Configure a MS SQL Server Data Source driver.*
3. *Configure a MS SQL Server Connection.*
4. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

### How to Configure a Microsoft SQL Server Data Source

Available in the Enterprise edition only.
The steps for configuring a data source for connecting to a Microsoft SQL server are as follows:

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

   The dialog box for configuring a data source is opened.

   ![Data Source Drivers Configuration Dialog Box](image)

   **Figure 208: Data Source Drivers Configuration Dialog Box**

3. Enter a unique name for the data source.
4. Select **SQLServer** in the driver **Type** drop-down list.
5. Add the Microsoft SQL Server driver file using the **Add Files** button.

   The SQL Server driver file is called `sqljdbc.jar`. In the **Driver files** section lists download links for database drivers that are necessary for accessing Microsoft SQL Server databases in Oxygen XML Developer plugin.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

### How to Configure a Microsoft SQL Server Connection

The support to configure a Microsoft SQL Server connection is available in the Enterprise edition only.

To configure a connection to a Microsoft SQL Server, follow these steps:

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.
3. Enter a unique name for the connection.
4. Select the SQL Server data source in the Data Source drop-down list.
5. Enter the connection details.
   a) Enter the URL of the SQL Server server.
      If you want to connect to the server using Windows integrated authentication, you must add
      ;integratedSecurity=true to the end of the URL. The URL will look like this:
      
      \[jdbc:sqlserver://localhost;instanceName=SQLEXPRESS;integratedSecurity=true;\]
      
      **Note:** For integrated authentication, leave the User and Password fields empty.
   b) Enter the user name for the connection to the SQL Server.
   c) Enter the password for the connection to the SQL Server.
6. Click the OK button to finish the configuration of the database connection.

**How to Configure Generic JDBC Support**

To configure the support for a generic JDBC database follow this procedure:

1. **Configure a Generic JDBC Data Source driver.**
2. **Configure a Generic JDBC Connection.**
3. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).

**How to Configure a Generic JDBC Data Source**

Starting with version 17, Oxygen XML Developer plugin comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Developer plugin along with a Java VM version 7 or 6. The following procedure shows you how to configure a generic JDBC data source:

1. **Open the Preferences dialog box** and go to Data Sources.
2. Click the ✦ New button in the Data Sources panel.
3. Enter a unique name for the data source.
4. Select *Generic JDBC* in the driver *Type* drop-down list.
5. Add the driver file(s) using the *Add Files* button.
6. Select the most appropriate *Driver class*.
7. Click the *OK* button to finish the data source configuration.

**How to Configure a Generic JDBC Connection**

To configure a connection to a generic JDBC database, follow these steps:

1. *Open the Preferences dialog box* and go to *Data Sources*.
2. In the *Connections* panel, click the *New* button.
3. Enter a unique name for the connection.
4. Select the *Generic JDBC* data source in the *Data Source* drop-down list.
5. Enter the connection details.
   a) Enter the URL of the generic JDBC database, with the following format: `jdbc: <subprotocol>: <subname>`.
   b) Enter the user name for the connection to the generic JDBC database.
   c) Enter the password for the connection to the generic JDBC database.
6. Click the *OK* button to finish the configuration of the database connection.

**How to Configure MySQL Support**

To configure the support for a MySQL database follow this procedure:

1. *Configure a MySQL Data Source driver*.
2. *Configure a MySQL Connection*.
3. Use the *Data Source Explorer* view from the *Window > Show View > Data Source Explorer* menu or switch to the *Database Perspective* (available from *Window > Open Perspective > Database*).

**How to Configure a MySQL Data Source**

To connect to a MySQL server, create a data source of a generic JDBC type, based on the *MySQL JDBC driver available on the MySQL website*. The following steps describe how you can configure such a data source:

1. *Open the Preferences dialog box* and go to *Data Sources*.
2. Click the *New* button in the *Data Sources* panel.
   The dialog box for configuring a data source is opened.
3. Enter a unique name for the data source.
4. Select *Generic JDBC* in the driver *Type* drop-down list.
5. Add the MySQL driver files using the *Add Files* button. The driver file for the MySQL server is called *mysql-com.jar*. The *Driver files* section lists *download links for database drivers* that are necessary for accessing MySQL databases in Oxygen XML Developer plugin.
6. Select the most appropriate *Driver class*.
7. Click the *OK* button to finish the data source configuration.

**How to Configure a MySQL Connection**

To configure a connection to a MySQL server, follow these steps:

1. *Open the Preferences dialog box* and go to *Data Sources*.
2. In the *Connections* panel, click the *New* button.
   
   The dialog box for configuring a database connection is displayed.
3. Enter a unique name for the connection.
4. Select the MySQL data source in the Data Source drop-down list.
5. Enter the connection details.
   a) Enter the URL of the MySQL server.
   b) Enter the user name for the connection to the MySQL server.
   c) Enter the password for the connection to the MySQL server.
6. Click the OK button to finish the configuration of the database connection.

How to Oracle 11g Support
To configure the support for a Oracle 11g database follow this procedure:

1. Go to the Oracle website and download the Oracle 11g JDBC driver called ojdbc6.jar.
2. Configure a Oracle 11g Data Source driver.
3. Configure a Oracle 11g Connection.
4. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).

How to Configure an Oracle 11g Data Source
Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an Oracle 11g server are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Data Sources panel.
   The dialog box for configuring a data source is opened.
3. Enter a unique name for the data source.
4. Select Oracle in the driver Type drop-down list.
5. Add the Oracle driver file using the Add Files button.
   The Oracle driver file is called ojdbc5.jar. The Driver files section lists download links for database drivers that are necessary for accessing Oracle databases in Oxygen XML Developer plugin.
6. Select the most appropriate Driver class.
7. Click the OK button to finish the data source configuration.

**How to Configure an Oracle 11g Connection**

Available in the Enterprise edition only.

The steps for configuring a connection to an Oracle 11g server are as follows:

1. *Open the Preferences dialog box* and go to Data Sources.
2. In the Connections panel, click the ➕ New button.
   The dialog box for configuring a database connection is displayed.
3. Enter a unique name for the connection.
4. Select the Oracle 11g data source in the Data Source drop-down list.
5. Enter the connection details.
   a) Enter the URL of the Oracle server.
   b) Enter the user name for the connection to the Oracle server.
   c) Enter the password for the connection to the Oracle server.
6. Click the OK button to finish the configuration of the database connection.

**How to Configure PostgreSQL Support**

To configure the support for a PostgreSQL database follow this procedure:

1. Go to the PostgreSQL website and download the PostgreSQL 8.3 JDBC driver called postgresql-8.3-603.jdbc3.jar.
2. Configure a PostgreSQL Data Source driver.
3. Configure a PostgreSQL Connection.
4. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).

**How to Configure a PostgreSQL 8.3 Data Source**

The steps for configuring a data source for connecting to a PostgreSQL server are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Data Sources panel.
   The dialog box for configuring a data source is opened.
3. Enter a unique name for the data source.
4. Select *PostgreSQL* in the driver *Type* drop-down list.
5. Add the PostgreSQL driver file using the Add Files button. The PostgreSQL driver file is called postgresql-8.3-603.jdbc3.jar. The Driver files section lists download links for database drivers that are necessary for accessing PostgreSQL databases in Oxygen XML Developer plugin.

6. Select the most appropriate Driver class.

7. Click the OK button to finish the data source configuration.

How to Configure a PostgreSQL 8.3 Connection

The steps for configuring a connection to a PostgreSQL 8.3 server are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. In the Connections panel, click the New button.

   The dialog box for configuring a database connection is displayed.

![Connection Configuration Dialog Box](image)

Figure 216: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select the PostgreSQL 8.3 data source in the Data Source drop-down list.
5. Enter the connection details.
   a) Enter the URL of the PostgreSQL 8.3 server.
   b) Enter the user name for the connection to the PostgreSQL 8.3 server.
   c) Enter the password for the connection to the PostgreSQL 8.3 server.
6. Click the OK button to finish the configuration of the database connection.

How to Configure JDBC-ODBC Support

To configure the support for a JDBC-ODBC database follow this procedure:

1. Configure a JDBC-ODBC Data Source driver.
2. Configure a JDBC-ODBC Connection.
3. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).
**How to Configure a JDBC-ODBC Connection**

Starting with version 17, Oxygen XML Developer plugin comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Developer plugin along with a Java VM version 7 or 6. To configure a connection to an ODBC data source, follow these steps:

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

   ![Connection Configuration Dialog Box](image)

3. Enter a unique name for the connection.
4. Select **JDBC-ODBC Bridge** in the **Data Source** drop-down list.
5. Enter the connection details.
   a) Enter the URL of the ODBC source.
   b) Enter the user name of the ODBC source.
   c) Enter the password of the ODBC source.
6. Click the **OK** button to finish the configuration of the database connection.

**Resource Management**

This section explains resource management actions for relational databases.

**Data Source Explorer View**

The **Data Source Explorer** view displays your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded to the column level. Oxygen XML Developer plugin supports multiple simultaneous database connections and the connection tree provides an easy method for browsing them.
The following objects are displayed in the **Data Source Explorer** view:

- Connection
- Collection (Catalog)
- XML Schema Repository
- XML Schema Component
- Schema
- Table
- System Table
- Table Column

A **Collection** (called *catalog* in some databases) is a hierarchical container for resources and sub-collections. There are two types of resources:

- **XML resource** - an XML document or document fragment, selected by a previously executed XPath query.
- **non-XML resource** - any resource that is not recognized as XML.

**Note:** For some connections you can add or move resources into a container by dragging them from:

- the **Project view**
- the default file system application (for example, Windows Explorer in Windows or Finder in Mac OS X)
- another database container

The following actions are available in the toolbar of this view:

**Filters**

Opens the **Data Sources / Table Filters** *Preferences page*, allowing you to decide which table types are displayed in the **Data Source Explorer** view.
Configure Database Sources
Opens the Data Sources preferences page where you can configure both data sources and connections.

Actions Available at Connection Level in Data Source Explorer View
The contextual menu of a Connection node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

- **Disconnect**
  Closes the current database connection. If a table is already open, you are warned to close it before proceeding.

- **Configure Database Sources**
  Opens the Data Sources preferences page where you can configure both data sources and connections.

Actions Available at Catalog Level in Data Source Explorer View
The contextual menu of a Catalog node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

Actions Available at Schema Level in Data Source Explorer View
The contextual menu of a Schema node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

Actions Available at Table Level in Data Source Explorer View
The contextual menu of a Table node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

- **Edit**
  Opens the selected table in the Table Explorer view.

- **Export to XML**
  Opens the Export Criteria dialog (a thorough description of this dialog can be found in the Import from Database chapter).

XML Schema Repository Level
This section explains the actions available at the XML Schema Repository level.

Oracle's XML Schema Repository Level
The Oracle database supports XML schema repository (XSR) in the database catalogs. The contextual menu of a XML Schema Repository node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

- **Register**
  Opens a dialog for adding a new schema file in the XML repository. To add an XML Schema, enter the schema URI and location on your file system. *Local* scope means that the schema is visible only to the user who registers it. *Global* scope means that the schema is public.
**Note:** Registering a schema may involve dropping/creating types. Hence you need type-related privileges such as DROP TYPE, CREATE TYPE, and ALTER TYPE. You need privileges to delete and register the XML schemas involved in the registering process. You need all privileges on XMLType tables that conform to the registered schemas. For XMLType columns, the ALTER TABLE privilege is needed on corresponding tables. If there are schema-based XMLType tables or columns in other database schemas, you need privileges such as the following:

- CREATE ANY TABLE
- CREATE ANY INDEX
- SELECT ANY TABLE
- UPDATE ANY TABLE
- INSERT ANY TABLE
- DELETE ANY TABLE
- DROP ANY TABLE
- ALTER ANY TABLE
- DROP ANY INDEX

To avoid having to grant all these privileges to the schema owner, Oracle recommends that the registration be performed by a DBA if there are XML schema-based XMLType table or columns in other user database schemas.

**IBM DB2's XML Schema Repository Level**

The contextual menu of a XML Schema Repository node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

- **Register**
  Opens a dialog box for adding a new schema file in the XML Schema repository. In this dialog box, the following fields can be set:
  - **XML schema file** - Location on your file system.
  - **XSR name** - Schema name.
  - **Comment** - Short comment (optional).
  - **Schema location** - Primary schema name (optional).

  Decomposition means that parts of the XML documents are stored in relational tables. Which parts map to which tables and columns are specified in the schema annotations. Schema dependencies management is done by using the Add and Remove buttons.

The actions available at Schema level are as follows:

- **Refresh**
  Performs a refresh of the selected node (and its sub-tree).

- **Unregister**
  Removes the selected schema from the XML Schema Repository.

- **View**
  Opens the selected schema in Oxygen XML Developer plugin.
Microsoft SQL Server’s XML Schema Repository Level

The contextual menu of a XML Schema Repository node in the tree from the Data Source Explorer view contains the following actions:

- **Refresh**
  Performs a refresh for the sub-tree of the selected node.

- **Register**
  Opens a dialog for adding a new schema file in the DB XML repository. In this dialog you enter a collection name and the necessary schema files. XML Schema files management is done by using the **Add** and **Remove** buttons.

The actions available at Schema level are as follows:

- **Refresh**
  Performs a refresh of the selected node (and its sub-tree).

- **Add**
  Adds a new schema to the XML Schema files.

- **Unregister**
  Removes the selected schema from the XML Schema Repository.

- **View**
  Opens the selected schema in Oxygen XML Developer plugin.

Table Explorer View

Every table from the Data Source Explorer view can be displayed and edited in the Table Explorer view by pressing the **Edit** button from the contextual menu or by double-clicking one of its fields. To modify the content of a cell, double-click it and start typing. When editing is complete, Oxygen XML Developer plugin attempts to update the database with the new cell content.

![Figure 219: The Table Explorer View](image)

You can sort the content of a table by one of its columns by clicking on its column header.

Note the following:

- The first column is an index (not part of the table structure)
- Every column header contains the field name and its data type
- The primary key columns are marked with this symbol: 🔒
- Multiple tables are presented in a tabbed manner
For performance issues, you can set the maximum number of cells that are displayed in the Table Explorer view (using the Limit the number of cells field from the Data Sources Preferences page). If a table that has more cells than the value set in the options is displayed in the Table Explorer view, a warning dialog informs you that the table is only partially shown.

You are notified if the value you have entered in a cell is not valid (and thus cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, an information dialog appears, notifying you that the value you have inserted cannot be converted to the SQL type of that field. For example, if you have a column that contains LONG (numerical) values, and a character or string is inserted into one of its cells, you would get the error message that a string value cannot be converted to the requested SQL type (NUMBER).

- If the constraints of the database are not met (for instance, primary key constraints), an information dialog will appear, notifying you of the reason the database has not been updated. For example, in the table below, trying to set the second record in the primary key propID column to 8, results in a duplicate entry error since that value has already been used in the first record:

![Figure 220: Duplicate Entry for Primary Key](image)

Common edit actions (Cut, Copy, Paste, Select All, Undo, Redo) are available in the popup menu of an edited cell.

The contextual menu, available on every cell, also has the following actions:

Set NULL
- Sets the content of the cell to null. This action is disabled for columns that cannot have a value of null.

Insert row
- Inserts an empty row in the table.

Duplicate row
- Makes a copy of the selected row and adds it in the Table Explorer view. Note that the new row will not be inserted in the database table until all conflicts are resolved.

Commit row
- Commits the selected row.

Delete row
- Deletes the selected row.

Copy
- Copies the content of the cell.
Paste
Paste copied content into the selected cell.

The Table Explorer toolbar also includes the following actions:

Export to XML
Opens the Export Criteria dialog (a thorough description of this dialog can be found in the Import from database chapter).

Refresh
Performs a refresh for the sub-tree of the selected node.

Insert row
Inserts an empty row in the table.

Duplicate row
Makes a copy of the selected row and adds it in the Table Explorer view. Note that the new row will not be inserted in the database table until all conflicts are resolved.

Commit row
Commits the selected row.

Delete row
Deletes the selected row.

SQL Execution Support
Oxygen XML Developer plugin’s support for writing SQL statements includes syntax highlighting, folding, and dragging and dropping from the Data Source Explorer view. It also includes transformation scenarios for executing the statements, and the results are displayed in the Table Explorer view.

Drag and Drop from Data Source Explorer View
Drag and drop operations from the Data Source Explorer view to the SQL Editor allows you to create SQL statements quickly by inserting the names of tables and columns in the SQL statements.

1. Configure a database connection (see the specific procedure for your database server).
2. Browse to the table you will use in your statement.
3. Drag the table or a column of the table into the editor where a SQL file is open.

Drag and drop actions are available both on the table and on its fields. A pop-up menu is displayed in the SQL editor.
4. Select the type of statement from the pop-up menu.

Depending on your choice, dragging a table results in one of the following statements being inserted into the document:

- **SELECT `field1`, `field2`, .... FROM `catalog`.`table`** (for example: SELECT `DEPT`, `DEPTNAME`, `LOCATION` FROM `camera`.`cameraDesc`)
- **UPDATE `catalog`.`table` SET `field1` = `field2` = ....** (for example: UPDATE `camera`.`cameraDesc` SET `DEPT` = `DEPTNAME` = `LOCATION` =)
- **INSERT INTO `catalog`.`table` (`field1`, `field2`, ....) VALUES (), ()** (for example: INSERT INTO `camera`.`cameraDesc` (`DEPT`, `DEPTNAME`, `LOCATION`) VALUES (), ()
- **DELETE FROM `catalog`.`table`** (for example: DELETE FROM `camera`.`cameraDesc`)

Depending on your choice, dragging a column results in one of the following statements being inserted into the document:

- **SELECT `field` FROM `catalog`.`table`** (for example: SELECT `DEPT` FROM `camera`.`cameraDesc`)
- **UPDATE `catalog`.`table` SET `field` =** (for example: UPDATE `camera`.`cameraDesc` SET `DEPT` =)
- **INSERT INTO `catalog`:`table` (`field1`) VALUES ()** (for example: INSERT INTO `camera`:`cameraDesc` (`DEPT`) VALUES ()
- **DELETE FROM `catalog`:`table`** (for example: DELETE FROM `camera`.`cameraDesc` WHERE `DEPT` =)

**SQL Validation**

SQL validation support is offered for IBM DB2. Please note that if you choose a connection that does not support SQL validation, you will receive a warning when trying to validate. The SQL document is validated using the connection from the associated transformation scenario.
Executing SQL Statements

The steps for executing an SQL statement on a relational database are as follows:

1. **Configure a transformation scenario** using the ![Configure Transformation Scenario(s)](Transformation) action from the Transformation toolbar or the XML menu.

   A SQL transformation scenario needs a database connection. You can configure a connection using the ![Preferences](Preferences) button from the SQL transformation dialog box.

   The dialog box contains the list of existing scenarios that apply to SQL documents.

2. Set parameter values for SQL placeholders using the Parameters button from the SQL transformation dialog box. For example, in `SELECT * FROM `test`.'department` where DEPT = ? or DEPTNAME = ? the two parameters can be configured for the place holders (?) in the transformation scenario.

   When the SQL statement is executed, the first placeholder is replaced with the value set for the first parameter in the scenario, the second placeholder is replaced by the second parameter value, and so on.

   **Restriction:** When a stored procedure is called in an SQL statement executed on an SQL Server database, mixing in-line parameter values with values specified using the Parameters button of the scenario dialog box is not recommended. This is due to a limitation of the SQL Server driver for Java applications. An example of stored procedure that is not recommended: `call dbo.Test(22, ?)`.

3. Execute the SQL scenario by clicking the OK or Apply associated button.

   The result of a SQL transformation is displayed in a view at the bottom of the Oxygen XML Developer plugin window.

4. View more complex return values of the SQL transformation in a separate editor panel.

   A more complex value returned by the SQL query (for example, an XMLTYPE or CLOB value) cannot be displayed entirely in the result table.

   a) Right-click on the cell containing the complex value.

   b) Select the action Copy cell from the contextual menu.

   The action copies the value in the clipboard.

   c) Paste the value into an appropriate editor.

   For example, you can paste the value in an opened XQuery editor panel of Oxygen XML Developer plugin.

Native XML Database (NXD) Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. Oxygen XML Developer plugin offers support for the following native XML databases:

- Berkeley DB XML
- eXist
- MarkLogic
- Documentum xDb (X-Hive/DB) 10
- Oracle XML DB

To watch our video demonstration about the integration between the XML native databases and Oxygen XML Developer plugin, go to [http://www.oxygenxml.com/demo/Author_Database_XML_Native.html](http://www.oxygenxml.com/demo/Author_Database_XML_Native.html).

Configuring Database Data Sources

This section describes the procedures for configuring the following native database data sources:

- Berkeley DB XML
- eXist
- MarkLogic
Configuring Database Connections

This section describes the procedures for configuring the connections for the following native databases:

- Berkeley DB XML
- eXist
- MarkLogic
- Documentum xDb (X-Hive/DB) 10

How to Configure Support for Native XML Databases

This section contains procedures about configuring the support for various native XML databases.

How to Configure Berkeley DB XML Support

Follow this procedure to configure the support for a Berkeley DB XML database:

1. Configure a Berkeley DB XML Data Source driver.
2. Configure a Berkeley DB XML Connection.
3. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).

How to Configure a Berkeley DB XML Data Source

Oxygen XML Developer plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a data source for a Berkeley DB XML database are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Data Sources panel.
3. Enter a unique name for the data source.
4. Select Berkeley DBXML from the driver Type drop-down list.
5. Click the Add button to add the Berkeley DB driver files.
   The driver files for the Berkeley DB database are the following:
   a) db.jar (check for it in [DBXML_DIR] / lib or [DBXML_DIR] / jar)
   b) dbxml.jar (check for it in [DBXML_DIR] / lib or [DBXML_DIR] / jar)

   Where [DBXML_DIR] is the Berkeley DB XML database root directory. For example, in Windows it is: C:\Program Files\Oracle\Berkeley DB XML <version>.
6. Click the OK button to finish the data source configuration.

How to Configure a Berkeley DB XML Connection

Oxygen XML Developer plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a connection to a Berkeley DB XML database are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Connections panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the Data Source drop-down list.
5. Enter the connection details.
   a) Set the path to the Berkeley DB XML database directory in the Environment home directory field. Use a directory with write access. DO NOT use the installation directory where Berkeley DB XML is installed if you do not have write access to that directory.
   b) Select the Verbosity level: DEBUG, INFO, WARNING, or ERROR.
   c) Optionally, you can select the check-box Join existing environment.
If checked, an attempt is made to join an existing environment in the specified home directory and all the original environment settings are preserved. If that fails, try reconfiguring the connection with this option unchecked.

6. Click the **OK** button to finish the connection configuration.

**How to Configure eXist Support**

Follow this procedure to configure the support for an eXist database:

1. **Configure a eXist Data Source driver.**
2. **Configure a eXist Connection.**
3. Use the Data Source Explorer view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

**How to Configure an eXist Data Source**

Oxygen XML Developer plugin supports eXist database server versions up to and including version 2.2. The steps for configuring a data source for an eXist database are as follows:

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select **eXist** from the driver **Type** drop-down list.
5. Click the **Add** button to add the eXist driver files.

   The following driver files should be added in the dialog box for setting up the eXist datasource. They are found in the installation directory of the eXist database server. Please make sure you copy the files from the installation of the eXist server where you want to connect from Oxygen XML Developer plugin.
   
   • exist.jar
   • lib/core/xmlmdb.jar
   • lib/core/xmlrpc-client-3.1.x.jar
   • lib/core/xmlrpc-common-3.1.x.jar
   • lib/core/ws-commons-util-1.0.x.jar
   • lib/core/slf4j-api-1.x.x.jar (if available)
   • lib/core/slf4j-log4j12-1.x.x.jar (if available)

   The version number from the driver file names may be different for your eXist server installation.

6. Click the **OK** button to finish the connection configuration.

To watch our video demonstration about running XQuery against an eXist XML database, go to [http://www.oxygenxml.com/demo/eXist_Database.html](http://www.oxygenxml.com/demo/eXist_Database.html).

**How to Configure an eXist Connection**

The steps for configuring a connection to an eXist database are as follows:

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down list.
5. Enter the connection details.
   a) Set the URI to the installed eXist engine in the **XML DB URI** field.
   b) Set the user name in the **User** field.
   c) Set the password in the **Password** field.
   d) Enter the start collection in the **Collection** field.

   eXist organizes all documents in hierarchical collections. Collections are like directories. They are used to group related documents together. This text field allows the user to set the default collection name.
6. Click the OK button to finish the connection configuration.

To watch our video demonstration about running XQuery against an eXist XML database, go to
http://www.oxygenxml.com/demo/eXist_Database.html.

The Create eXist-db XML Connection Dialog Box

A quick way to create an eXist connection is to use the dedicated Create eXist-db XML connection dialog box. Open the Preferences dialog box, go to Data Sources and click Create eXist-db XML connection. After you fill in the fields, click OK and go to Window > Show View > Data Source Explorer to view your connection.

To create an eXist connection using this dialog box, Oxygen XML Developer plugin expects the exist/webstart/exist.jnlp path to be accessible at the provided Host and Port.

How to Configure MarkLogic Support

Follow this procedure to configure the support for a MarkLogic database:

1. Download the MarkLogic driver from MarkLogic Community site.
2. Configure a MarkLogic Data Source driver.
3. Configure a MarkLogic Connection.
4. Use the Data Source Explorer view from the Window > Show View > Data Source Explorer menu or switch to the Database Perspective (available from Window > Open Perspective > Database).

How to Configure a MarkLogic Data Source

Available in the Enterprise edition only.

Note: Oxygen XML Developer plugin supports MarkLogic version 4.0 or later.

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Data Sources panel.
3. Enter a unique name for the data source.
4. Select MarkLogic from the driver Type drop-down list.
5. Click the Add button to add the MarkLogic driver file (marklogic-xcc-{server_version}, where {server_version} is the MarkLogic server version.)

You can download the driver file from: http://community.marklogic.com/download.

6. Click the OK button to finish the data source configuration.

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.

Note: Oxygen XML Developer plugin supports MarkLogic version 4.0 or later.

The steps for configuring a connection to a MarkLogic database are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. Click the New button in the Connections panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the Data Source drop-down list.
5. Enter the connection details.

a) The host name or IP address of the installed MarkLogic engine in the XDBC Host field.

Oxygen XML Developer plugin uses XCC connector to interact with MarkLogic XDBC server and requires the basic authentication schema to be set. Starting with version MarkLogic 4.0 the default authentication method when you create a HTTP or WebDAV Server is digest, so make sure to change it to basic.

b) Set the port number of the MarkLogic engine in the Port field. A MarkLogic XDBC application server must be configured on the server on this port. This XDBC server will be used to execute XQuery expressions against the
server. Later, if you want to change the XDBC server, instead of editing the configuration just use the *Use it to execute queries* action from Data Source Explorer.

c) Set the user name to access the MarkLogic engine in the **User** field.
d) Set the password to access the MarkLogic engine in the **Password** field.
e) Optionally set the URL used for browsing the MarkLogic database in the **Data Source Explorer** view in the **WebDAV URL** field.

The **Database** field specifies the database over which the XQuery expressions are executed. If you set this option to default, the database associated to the application server of the configured port is used.

6. Click the **OK** button to finish the connection configuration.

**How to Configure Documentum xDb (X-Hive/DB) 10 Support**

Follow this procedure to configure the support for a Documentum xDb (X-Hive/DB) 10 database:

1. **Configure a Documentum xDb Data Source driver.**
2. **Configure a Documentum xDb Connection.**
3. Use the **Data Source Explorer** view from the **Window > Show View > Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

**How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source**

Available in the Enterprise edition only.

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select **XHive** from the driver **Type** drop-down list.
5. Click the **Add** button to add the XHive driver files.

The driver files for the Documentum xDb (X-Hive/DB) 10 database are found in the Documentum xDb (X-Hive/DB) 10 lib directory from the server installation folder:

- antlr-runtime.jar
- aspectjrt.jar
- icu4j.jar
- xhive.jar
- google-collect.jar

6. Click the **OK** button to finish the data source configuration.

**How to Configure an Documentum xDb (X-Hive/DB) 10 Connection**

The steps for configuring a connection to a Documentum xDb (X-Hive/DB) 10 database are as follows:

- **Note:** The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Developer plugin. The following procedure explains the xhive:// protocol connection type.

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down list.
5. Enter the connection details.

   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 Run XQuery in read / write session (with committing) checkbox if you want to end the session with a commit. Otherwise, the session ends with a rollback.

6. Click the OK button to finish the connection configuration.

Data Source Explorer View

The Data Source Explorer view displays your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded to the column level. supports multiple simultaneous database connections and the connection tree provides an easy method for browsing them.

![Data Source Explorer View](image)

Figure 222: Data Source Explorer View

The following objects are displayed in the Data Source Explorer view:

- Connection
- Collection (Catalog)
- XML Schema Repository
- XML Schema Component
- Schema
- Table
- System Table
- Table Column

A Collection (called catalog in some databases) is a hierarchical container for resources and sub-collections. There are two types of resources:

- XML resource - an XML document or document fragment, selected by a previously executed XPath query.
- non-XML resource - any resource that is not recognized as XML.
Note: For some connections you can add or move resources into a container by dragging them from:

- the Project view
- the default file system application (for example, Windows Explorer in Windows or Finder in Mac OS X)
- another database container

The following actions are available in the toolbar of this view:

Filters
Opens the Data Sources / Table Filters Preferences page, allowing you to decide which table types are displayed in the Data Source Explorer view.

Configure Database Sources
Opens the Data Sources preferences page where you can configure both data sources and connections.

Oracle XML DB Browser
Oracle XML DB is a feature of the Oracle database. It provides a high-performance, native XML storage and retrieval technology. Oxygen XML Developer plugin allows you to browse the native Oracle XML Repository and perform various operations on the resources in the repository.

![Data Source Explorer](image)

Figure 223: Browsing the Oracle XML DB Repository

The actions available at XML Repository level are as follows:

Refresh
Performs a refresh of the XML Repository.

Add container
Adds a new child container to the XML Repository.

Add resource
Adds a new resource to the XML Repository.

The actions available at container level are as follows:

Refresh
Performs a refresh of the selected container.

Add container
Adds a new child container to the current one.

Add resource
Adds a new resource to the folder.
Delete
  Deletes the current container.

Properties
  Shows various properties of the current container.

The actions available at resource level are as follows:

- **Refresh**
  Performs a refresh of the selected resource.

- **Open**
  Opens the selected resource in the editor.

- **Rename**
  Renames the current resource

- **Move**
  Moves the current resource to a new container (also available through drag and drop).

- **Delete**
  Deletes the current resource.

- **Copy location**
  Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

- **Properties**
  Shows various properties of the current resource.

- **Compare**
  Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

For running an XQuery transformation on collections from the XML Repository, please see a tutorial from Oracle.

**PostgreSQL Connection**

Oxygen XML Developer plugin allows you to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.
The actions available at container level are as follows:

- **Refresh**
  Performs a refresh of the selected container.

The actions available at resource level are as follows:

- **Refresh**
  Performs a refresh of the selected database table.

- **Edit**
  Opens the selected database table in the **Table Explorer** view.

- **Export to XML ...**
  Exports the content of the selected database table as an XML file using *the dialog from importing data from a database*.

- **Compare**
  Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

**Berkeley DB XML Connection**

This section explains the actions that are available on a Berkeley DB XML connection.

**Actions Available at Connection Level**

In a Berkeley DB XML repository, the actions available at connection level in the **Data Source Explorer** view are as follows:
**Refresh**  
Performs a refresh for the sub-tree of the selected node.

**Disconnect**  
Closes the current database connection.

**Configure Database Sources**  
Opens the *Data Sources preferences page* where you can configure both data sources and connections.

**Add container**  
Adds a new container in the repository with the following attributes:

- **Name** - The name of the new container.
- **Container type** - At creation time, every container must have a type defined for it. This container type identifies how XML documents are stored in the container. As such, the container type can only be determined at container creation time. You cannot change it when subsequent container opens. Containers can have one of the following types specified for them:
  - **Node container** - XML documents are stored as individual nodes in the container. Each record in the underlying database contains a single leaf node, its attributes and attribute values (if any), and its text nodes (if any). Berkeley DB XML also keeps the information it requires to reassemble the document from the individual nodes stored in the underlying databases. This is the default, and preferred, container type.
  - **Whole document container** - The container contains entire documents. The documents are stored without any manipulation of line breaks or whitespace.
  - **Allow validation** - If checked, it causes documents to be validated when they are loaded into the container. The default behavior is to not validate documents.
  - **Index nodes** - If checked, it causes indices for the container to return nodes rather than documents. The default is to index at the document level. This property has no meaning if the container type is Whole document container.

**Properties**  
Shows a dialog box that contains a list of the Berkeley connection properties (*version*, *home location*, *default container type*, *compression algorithm*, etc.)

**Actions Available at Container Level**

In a Berkeley DB XML repository, the actions available at container level in the *Data Source Explorer* view are as follows:

**Add Resource**  
Adds a new XML resource to the selected container.

**Rename**  
Allows you to specify a new name for the selected container.

**Delete**  
Removes the selected container from the database tree.

**Edit indices**  
Allows you to edit the indices for the selected container.

**Refresh**  
Performs a refresh for the sub-tree of the selected node.
Properties

Displays a dialog box with a list of properties of the Berkeley container (such as container type, auto indexing, page size, validate on load, compression algorithm, number of documents, etc.)

Figure 225: Container indices

The fields of the dialog box are as follows:

- Granularity:
  - Document level - Good option for retrieving large documents.
  - Node level - Good option for retrieving nodes from within documents.

- Add / Edit indices:
  - Node - The node name.
  - Namespace - The index namespace.

- Index strategy:
  - Index type:
    - Uniqueness - Indicates whether the indexed value must be unique within the container.
  - Path type:
    - node - Indicates that you want to index a single node in the path.
    - edge - Indicates that you want to index the portion of the path where two nodes meet.

- Node type:
  - element - An element node in the document content.
  - attribute - An attribute node in the document content.
• **metadata** - A node found only in the metadata content of a document.

• **Key type:**
  - **equality** - Improves the performances of tests that look for nodes with a specific value.
  - **presence** - Improves the performances of tests that look for the existence of a node regardless of its value.
  - **substring** - Improves the performance of tests that look for a node whose value contains a given sub-string.

• **Syntax types** - The syntax describes the type of data the index contains and is mostly used to determine how indexed values are compared.

### Actions Available at Resource Level

In a Berkeley DB XML repository, the actions available at resource level in the **Data Source Explorer** view are as follows:

- **Refresh**
  - Performs a refresh of the selected resource.
- **Open**
  - Opens the selected resource in the editor.
- **Rename**
  - Allows you to change the name of the selected resource.
- **Move**
  - Allows you to move the selected resource in a different container in the database tree (also available through drag and drop).
- **Delete**
  - Removes the selected resource from the container.
- **Copy location**
  - Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.
- **Compare**
  - Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

### eXist Connection

This section explains the actions that are available on an eXist connection.

### Actions Available at Connection Level

For an eXist database, the actions available at connection level in the **Data Source Explorer** view are as follows:

- **Configure Database Sources**
  - Opens the Data Sources preferences page where you can configure both data sources and connections.
- **Disconnect**
  - Closes the current database connection.
- **Refresh**
  - Performs a refresh for the sub-tree of the selected node.

### Actions Available at Container Level

For an eXist database, the actions available at container level in the **Data Source Explorer** view are as follows:
New File
   Creates a file in the selected container.

New Collection
   Creates a collection.

Import Folders
   Adds the content of specified folders from the local file system.

Import Files
   Adds a set of XML resources from the local file system.

Cut
   Cuts the selected containers.

Copy
   Copies the selected containers.

   Note: You can add or move resources into the container by dragging them from the Project view, the default file management application (for example, Windows Explorer on Windows or Finder on OS X), or from another database container.

Paste
   Paste resources into the selected container.

Rename
   Allows you to change the name of the selected collection.

Delete
   Removes the selected collection.

Refresh
   Performs a refresh of the selected container.

Properties
   Allows you to view various useful properties associated with the container, such as name, creation date, owner, group, or permissions.

Actions Available at Resource Level

For an eXist database, the actions available at resource level in the Data Source Explorer view are as follows:

Refresh
   Performs a refresh of the selected resource.

Open
   Opens the selected resource in the editor.

Rename
   Allows you to change the name of the selected resource.

Cut
   Cuts the selected resources.

Copy
   Copies the selected resources.

   Note: You can add or move resources into the container by dragging them from the Project view, the default file management application (for example, Windows Explorer on Windows or Finder on OS X), or from another database container.

Paste
   Pastes the copied resources.
Delete
Removes the selected resource from the collection.

Copy location
Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

Properties
Allows you to view various useful properties associated with the resource.

Save As
Allows you to save the name of the selected binary resource as a file on disk.

Compare
Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

MarkLogic Connection
Once you configure a MarkLogic connection, you can use the Data Source Explorer view to display all the application servers that are configured on the server. You can expand each application server and view all the modules that it is configured to use. The Data Source Explorer view allows you to open and edit these modules.

Note: To browse modules located in a database, directory properties must be associated with them. These directory properties are generated automatically if the directory creation property of the database is set to automatic. If this property is set to manual or manual-enforced, add the directory properties of the modules manually, using the XQuery function xdmp:directory-create().

Manually Adding Directory Properties
For two documents with the /code/modules/main.xqy and /code/modules IMPORTS/import.xqy IDs, run this query:
(xdmp:directory-create(''/code/modules/''),
 xdmp:directory-create(''/code/modules/imports/'').

For further information about directory properties go to: http://blakeley.com/blogfile/2012/03/19/directory-assistance/.

When you execute or debug XQuery files opened from this view, the imported modules are better identified by the MarkLogic server. In a module, you are also able to add breakpoints that the debugger takes into account.

Note: Add breakpoints in the modules of the application server that executes the debugging.

Note: Open XQuery modules from the application server involved in the debugging or execution process.

In the Requests container of each application server, Oxygen XML Developer plugin displays both the queries that are stopped for debugging purposes and the queries that are still running. To clean up the entire Requests container at the end of your session, right-click it and use the Cancel all running requests action.
The **Data Source Explorer** view displays all the application servers available on the MarkLogic server. To change the XDBC application server that Oxygen XML Developer plugin uses to execute XQuery expressions, select the **Use it to execute queries** option from its contextual menu.

To manage resources for a MarkLogic database through WebDAV, configure a WebDAV URL in *the MarkLogic connection*.

The following actions are available in the contextual menu of the WebDAV connection:

- **Connection level actions:**
  - **Configure Database Sources...**
    Opens the **Data Sources preferences page**. Here you can configure both data sources and connections.
  - **New Folder...**
    Creates a new folder on the server.
  - **Import Files...**
    Allows you to add a new file on the server.
  - **Refresh**
    Performs a refresh of the connection.
  - **Find/Replace in Files...**
    Allows you to find and replace text in multiple files from the server.
• Folder level actions:
  
  **New File**
  Creates a new file on the server in the current folder.

  **New Folder...**
  Creates a new folder on the server.

  **Import Folders...**
  Imports folders on the server.

  **Import Files**
  Allows you to add a new file on the server in the current folder.

  **Cut**
  Removes the current selection and places it in the clipboard.

  **Copy**
  Copies the current selection into the clipboard.

  **Rename**
  Allows you to change the name of the selected folder.

  **Delete**
  Removes the selected folder.

  **Refresh**
  Refreshes the sub-tree of the selected node.

  **Find/Replace in Files...**
  Allows you to find and replace text in multiple files from the server.

• File level actions:

  **Open**
  Allows you to open the selected file in the editor.

  **Cut**
  Removes the current selection and places it in the clipboard.

  **Copy**
  Copies the current selection into the clipboard.

  **Copy Location**
  Copies an application-specific URL for the selected resource into the clipboard. You can use this URL for various actions, such as opening or transforming the resources.

  **Rename**
  Allows you to change the name of the selected file.

  **Delete**
  Removes the selected file.

  **Refresh**
  Performs a refresh of the selected node.

  **Properties**
  Displays the properties of the current file in a Properties dialog box.

  **Find/Replace in Files...**
  Allows you to find and replace text in multiple files from the server.
Compare
Compared the resources using Diff Files (this action is available in the contextual menu of two selected resources).

Documentum xDb (X-Hive/DB) Connections
This section explains the actions that are available on a Documentum xDb (X-Hive/DB) 10 connection.

Actions Available at Connection Level
For a Documentum xDb (X-Hive/DB) 10 database, the actions available at connection level in the Data Source Explorer view are as follows:

- **Refresh**: Performs a refresh for the subtree of the selected node.
- **Disconnect**: Closes the current database connection.
- **Configure Database Sources**: Opens the Data Sources preferences page where you can configure both data sources and connections.
- **Add library**: Allows you to add a new library.

**Insert XML Instance**: Allows you to add a new XML resource directly into the database root. See Documentum xDb (X-Hive/DB) 10 Parser Configuration for more details.

**Insert non-XML Instance**: Allows you to add a new non-XML resource directly in the database root.

**Properties**: Displays the connection properties.

Actions Available at Catalog Level
For a Documentum xDb (X-Hive/DB) 10 database, the actions available at catalog level in the Data Source Explorer view are as follows:

- **Refresh**: Performs a refresh of the selected catalog.
- **Add as models**: Allows you to add a new abstract schema model to the selected catalog.
- **Set default schema**: Allows you to set a default DTD to be used for parsing. It is not possible to set a default XML Schema.
- **Clear default schema**: Allows you to clear the default DTD. The action is available only if there is a DTD set as default.

**Properties**: Displays the catalog properties.

Actions Available at Schema Resource Level
For a Documentum xDb (X-Hive/DB) 10 database, the actions available at schema resource level in the Data Source Explorer view are as follows:

- **Refresh**: Performs a refresh of the selected schema resource.
- **Open**: Opens the selected schema resource in the editor.
**Rename**
Allows you to change the name of the selected schema resource.

**Save As**
Allows you to save the selected schema resource as a file on disk.

* **Delete**
Removes the selected schema resource from the catalog.

**Copy location**
Allows you to copy the URL of the selected schema resource to the clipboard.

**Set default schema**
Allows you to set the selected DTD to be used as default for parsing. The action is available only for DTD.

**Clear default schema**
Allows you to unset the selected DTD. The action is available only if the selected DTD is the current default to be used for parsing.

### Actions Available at Library Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at library level in the **Data Source Explorer** view are as follows:

- **Refresh**
  Performs a refresh of the selected library.

- **Add library**
  Adds a new library as a child of the selected library.

- **Add local catalog**
  Adds a catalog to the selected library. By default, only the root-library has a catalog, and all models are stored there.

- **Insert XML Instance**
  Allows you to add a new XML resource to the selected library. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.

- **Insert non-XML Instance**
  Allows you to add a new non-XML resource to the selected library.

- **Rename**
  Allows you to specify a new name for the selected library.

- **Move**
  Allows you to move the selected library to a different one (also available through drag and drop).

* **Delete**
  Removes the selected library.

- **Properties**
  Displays the library properties.

### Actions Available at Resource Level

When an XML instance document is added for a Documentum xDb (X-Hive/DB) 10 database, the actions available at resource level in the **Data Source Explorer** view are as follows:

- **Refresh**
  Performs a refresh of the selected resource.

- **Open**
  Opens the selected resource in the editor.

- **Rename**
  Allows you to change the name of the selected resource.
Move
Allows you to move the selected resource into a different library in the database tree (also available through drag and drop).

Note: You can copy or move resources by dragging them from another database catalog.

Save As
Allows you to save the selected binary resource as a file on disk.

× Delete
Removes the selected resource from the library.

Copy location
Allows you to copy the URL of the selected resource to the clipboard.

Add AS model
Allows you to add an XML schema to the selected XML resource.

Set AS model
Allows you to set an active AS model for the selected XML resource.

Clear AS model
Allows you to clear the active AS model of the selected XML resource.

Properties
Displays the resource properties. Available only for XML resources.

Compare
Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

Validation of an XML resource stored in a Documentum xDb (X-Hive/DB) 10 database is done against the schema associated with the resource in the database.

Documentum xDb (X-Hive/DB) 10 Parser Configuration for Adding XML Instances
When an XML instance document is added to a Documentum xDb (X-Hive/DB) 10 connection or library, it is parsed with an internal XML parser of the database server. The following options are available for configuring this parser:

• DOM Level 3 parser configuration parameters. More about each parameter can be found here: DOM Level 3 Configuration.

• Documentum xDb (X-Hive/DB) 10 specific parser parameters (for more information please consult the Documentum xDb (X-Hive/DB) 10 manual):
  
  • xhive-store-schema - If checked, the corresponding DTD or XML schemas are stored in the catalog during validated parsing.
  
  • xhive-store-schema-only-internal-subset - Stores only the internal sub-set of the document (not an external sub-set). This option modifies the xhive-store-schema (only has a function when that parameter is set to true, and when a DTD is involved). Select this option if you only want to store the internal sub-set of the document (not the external sub-set).
  
  • xhive-ignore-catalog - Ignores the corresponding DTD and XML schemas in the catalog during validated parsing.
  
  • xhive-psvi - Stores psvi information on elements and attributes. Documents parsed with this feature turned on give access to psvi information and enable support of data types by XQuery queries.
  
  • xhive-sync-features - With this convenience setting turned on, parameter settings of XhiveDocumentIf are synchronized with the parameter settings of LSParser. Note that parameter settings xhive-psvi and schema-location are always synchronized.
Troubleshooting

Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDVFFactoryImpl` does not extend from `DTDVFFactory`.

I am able to access my XML Database in the Data Source Explorer and open files for reading but when I try to save changes to a file back into the database, I receive the following error: "Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDVFFactoryImpl` does not extend from `DTDVFFactory`." How can I fix this?

Answer:

`xhive.jar` contains a MANIFEST.MF with a classpath:

```
Class-Path: core/antlr-runtime.jar core/aspectjrt.jar core/fastutil-shrinked.jar
  core/google-collect.jar core/icu4j.jar core/lucene-regex.jar core/lucene.jar
  core/serializer.jar core/xalan.jar core/xercesImpl.jar
```

Because the driver was configured to use `xhive.jar` directly from the `xDB` installation (where many other jars are located), `core/xercesImpl.jar` from the `xDB` installation directory is loaded even though it is not specified in the list of jars from the data source driver configuration (it is in the classpath from `xhive.jar`'s MANIFEST.MF). A simple workaround for this issue is to copy **ONLY** the jar files used in the driver configuration to a separate folder and configure the data source driver to use them from there.

XQuery and Databases

XQuery is a native XML query language that is useful for querying XML views of relational data to create XML results. It also provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data. The following database systems supported in Oxygen XML Developer plugin offer XQuery support:

- **Native XML Databases**:
  - Berkeley DB XML
  - eXist
  - MarkLogic (validation support available starting with version 5)
  - Documentum xDb (X-Hive/DB) 10

- **Relational Databases**:
  - IBM DB2
  - Microsoft SQL Server (validation support not available)
  - Oracle (validation support not available)

Build Queries with Drag and Drop from the Data Source Explorer View

When a query is edited in the XQuery editor, the XPath expressions can be composed quickly by dragging them from the Data Source Explorer view and dropping them into the editor panel.

1. Configure the data source to the relational database.
2. Configure the connection to the relational database.
3. Browse the connection in the Data Source Explorer view, expanded to the table or column that you want to insert in the query.
4. Drag the table or column name to the XQuery editor panel.
5. Drop the table or column name where the XPath expression is needed.

An XPath expression that selects the dragged name is inserted in the XQuery document at the caret position.
XQuery Transformation

XQuery is designed to retrieve and interpret XML data from any source, whether it is a database or document. Data is stored in relational databases but it is often required that the data be extracted and transformed as XML when interfacing to other components and services. Also, it is an XPath-based querying language supported by most NXD vendors. To perform a query, you need an XQuery transformation scenario.

1. Configure a data source for the database.
   The data source can be relational or XML native.

2. Configure an XQuery transformation scenario.
   a) Click the Configure Transformation Scenario toolbar button or go to menu Document > Transformation > Configure Transformation Scenario.
      The Configure Transformation Scenario dialog box is opened.

   b) Click the New button in the dialog box.
      The dialog box for editing an XQuery scenario is opened.
c) Insert the scenario name in the dialog box for editing the scenario.

d) Choose the database connection in the **Transformer** drop-down list.

e) Configure any other parameters as needed.

For an XQuery transformation, the output tab has an option called **Sequence** that allows you to execute an XQuery in lazy mode. The amount of data extracted from the database is controlled from the option **Size limit on Sequence view**. If you choose **Perform FO Processing** in the **FO Processor** tab, the **Sequence** option is ignored.

f) Click the **OK** button to finish editing the scenario.

Once the scenario is associated with the XQuery file, the query can include calls to specific XQuery functions that are implemented by that engine. The available functions depend on the target database engine selected in the scenario. For example, for eXist and Berkeley DB XML, **the Content Completion Assistant** lists the functions supported by that database engine. This is useful for only inserting calls to the supported functions (standard XQuery functions or extension ones) into the query.

**Note:** An XQuery transformation is executed against a Berkeley DB XML server as a transaction using the query transaction support of the server.

3. Run the scenario.

To view a more complex value returned by the query that cannot be entirely displayed in the XQuery query result table at the bottom of the Oxygen XML Developer plugin window (for example, an XMLTYPE or CLOB value), do the following:

- Right-click on that table cell.
- Select the **Copy cell** action from the pop-up menu to copy the value into the clipboard.
- Paste the value wherever you need it (for example, in an opened XQuery editor panel of Oxygen XML Developer plugin).
XQuery Database Debugging

This section describes the procedures for debugging XQuery transformations that are executed against MarkLogic and Berkeley DB XML databases.

Debugging with MarkLogic

To start a debug session against the MarkLogic engine, configure a MarkLogic data source and a MarkLogic connection. Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Developer plugin accesses. On the server side, debugging must be activated in the XDBC server and in the Task Server section of the server control console (the switch debug allow). If the debugging is not activated, the MarkLogic server reports the DBG–TASKDEBUGALLOW error.

The MarkLogic XQuery debugger integrates seamlessly into the XQuery Debugger perspective. If you have a MarkLogic scenario configured for the XQuery file, you can choose to debug the scenario directly. If not, switch to the XQuery Debugger perspective, open the XQuery file in the editor, and select the MarkLogic connection in the XQuery engine selector from the debug control toolbar. For general information about how a debugging session is started and controlled see the Working with the Debugger section.

If you want to debug an XQuery file stored on the MarkLogic server, we recommend you to use the Data Source Explorer view to open the module and start the debugging session. This improves the resolving of any imported modules.

Before starting a debugging session, we recommend that you link the MarkLogic connection with an Eclipse project. To do this, go to the Data Source Explorer view and select Link to project in the contextual menu of the MarkLogic connection. The major benefit of linking a debugging session with a project is that you can add breakpoints in the XQuery modules stored on the server. You are also able to access these modules from the Eclipse navigator and run debugging sessions from them.

Oxygen XML Developer plugin supports collaborative debugging. This feature allows multiple users to participate in the same debugging session. You can start a debugging session and at a certain point another user can continue it.

In a MarkLogic debugging session, when you add a breakpoint on a line where the debugger never stops, Oxygen XML Developer plugin displays a warning message. These warnings are displayed for breakpoints you add either in the main XQuery (which you can open locally or from the server), or for breakpoints you add in any XQuery that is opened from the connection that participates in the debugging session.

To watch our video demonstration about the XQuery debugger for MarkLogic, go to http://oxygenxml.com/demo/XQueryDebuggerforMarkLogic.html.

Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is available only for MarkLogic server versions 4.0 or newer.
- For MarkLogic server versions 4.0 or newer, there are three XQuery syntaxes that are supported: '0.9-ml' (inherited from MarkLogic 3.2), '1.0-ml', and '1.0'.
- The local debugger user interface presents all the debugging steps that the MarkLogic server executes and the results or possible errors of each step.
- All declared variables are presented as strings. The Value column of the Variables view contains the expression from the variable declaration. It can be evaluated by copying the expression with the Copy value action from the contextual menu of the Variables view and pasting it in the XWatch view.
- There is no support for Output to Source Mapping.
- There is no support for showing the trace.
- You can set Breakpoints in imported modules in one of the following cases:
  - when you open the module from the context of the application server involved in the debugging, using the data source explorer
  - when the debugger automatically opens the modules in the Editor
- No breakpoints are set in modules from the same server that are not involved in the current debugging session.
- No support for profiling when an XQuery transformation is executed in the debugger.
Debugging Queries Which Import Modules

When debugging queries on a MarkLogic database that imports modules stored in the database, the recommended steps for placing a breakpoint in a module are as follows:

1. Start the debugging session with the action `Debug Scenario` from the `Transformation` toolbar or the `XQuery Debugger` toolbar button.
2. Click `Step into` repeatedly until reaching the desired module.
3. Add the module to the current `project` for easy access.
4. Set breakpoints in the module as needed.
5. Continue debugging the query.

When starting a new debugging session, make sure that the modules that you will debug are already opened in the editor. This is necessary so that the breakpoints in the modules will be considered. Also, make sure that there are no other opened modules that are not involved in the current debugging session.

Debugging with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is supported in the Oxygen XML Developer plugin XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic debugger.

WebDAV Connection

This section explains how to work with a WebDAV connection in the Data Source Explorer view.

How to Configure a WebDAV Connection

By default, Oxygen XML Developer plugin is configured to contain a WebDAV data source connection called WebDAV (S)FTP. Based on this data source, you can create a WebDAV connection for browsing and editing data from a database that provides a WebDAV interface. The connection is available in the Data Source Explorer view. The steps for configuring a WebDAV connection are as follows:

1. Open the Preferences dialog box and go to Data Sources.
2. In the Connections panel, click the New button.
3. Enter a unique name for the connection.
4. Select one of the WebDAV data sources in the Data Source drop-down list.
5. Enter the connection details:
   a) Set the URL to the WebDAV repository in the field `WebDAV URL`.
   b) Set the user name that is used to access the WebDAV repository in the `User` field.
   c) Set the password that is used to access the WebDAV repository in the `Password` field.
6. Click the OK button.

To watch our video demonstration about the WebDAV support in Oxygen XML Developer plugin, go to [http://www.oxygenxml.com/demo/WebDAV_Support.html](http://www.oxygenxml.com/demo/WebDAV_Support.html).

WebDAV Connection Actions

This section explains the actions that are available for a WebDAV connection in the Data Source Explorer view.

Actions Available at Connection Level

The contextual menu of a WebDAV connection in the Data Source Explorer view contains the following actions:

- Configure Database Sources...

  Opens the Data Sources preferences page. Here you can configure both data sources and connections.
Disconnect
  Stops the connection.

☑️ Import Files...
  Allows you to add a new file on the server.

New Folder...
  Creates a new folder on the server.

🔄 Refresh
  Performs a refresh of the connection.

🔍 Find/Replace in Files...
  Allows you to find and replace text in multiple files from the server.

Actions Available at Folder Level
The contextual menu of a folder node in a WebDAV connection in the Data Source Explorer view contains the following actions:

New File
  Creates a new file on the server in the current folder.

New Folder...
  Creates a new folder on the server.

Import Folders...
  Imports folders on the server.

✓ Import Files
  Allows you to add a new file on the server in the current folder.

✂️ Cut
  Removes the current selection and places it in the clipboard.

коп    Copy
  Copies the current selection into the clipboard.

📋 Paste
  Pastes the copied selection.

Rename
  Allows you to change the name of the selected folder.

✗ Delete
  Removes the selected folder.

🔄 Refresh
  Refreshes the sub-tree of the selected node.

🔍 Find/Replace in Files...
  Allows you to find and replace text in multiple files from the server.

Actions Available at File Level
The contextual menu of a file node in a WebDAV connection in the Data Source Explorer view contains the following actions:

✓ Open
  Allows you to open the selected file in the editor.
Cut
Removes the current selection and places it in the clipboard.

Copy
Copies the current selection into the clipboard.

Copy Location
Copies an application-specific URL for the selected resource into the clipboard. You can use this URL for various actions, such as opening or transforming the resources.

Rename
Allows you to change the name of the selected file.

Delete
Removes the selected file.

Refresh
Performs a refresh of the selected node.

Properties
Displays the properties of the current file in a Properties dialog box.

Find/Replace in Files...
Allows you to find and replace text in multiple files from the server.

BaseX Support

This section explains how to configure the BaseX XML database support. The BaseX support is composed of two parts:

- Resource management in the Data Source Explorer view.
- XQuery execution.

Resource Management

Resource management is available by creating a WebDAV connection to the BaseX server.

First of all, make sure the BaseX HTTP Server is started. For details about starting the BaseX HTTP server, go to http://docs.basex.org/wiki/Startup#BaseX_HTTP_Server. The configuration file for the HTTP server is named .basex and is located in the BaseX installation directory. This file helps you to find out the port on which the HTTP server is running. The default port for BaseX WebDAV is 8984.

To ensure that everything is functioning, open a WebDAV URL inside a browser and check to see if it works. For example, the following URL retrieves a document from a database named TEST:

Once you are sure that the BaseX WebDAV service is working, you can configure the WebDAV connection in Oxygen XML Developer plugin as described in How to Configure a WebDAV Connection on page 453. The WebDAV URL should resemble this: http://{hostname}:{port}/webdav/. If the BaseX server is running on your own machine and it has the default configuration, the data required by the WebDAV connection is:

- WebDAV URL: http://localhost:8984/webdav
- User: admin
- Password: admin

Once the WebDAV connection is created, you can start browsing using the Data Source Explorer view.
**XQuery Execution**

XQuery execution is possible through an XQJ connection.

**BaseX XQJ Data Source**

First of all, create an XQJ data source as described in *How to Configure an XQJ Data Source* on page 377. The BaseX XQJ API-specific files that must be added in the configuration dialog are `xqj-api-1.0.jar`, `xqj2-0.1.0.jar` and `basex-xqj-1.2.3.jar` (the version names of the JAR file may differ). These libraries can be downloaded from `xqj.net/basex/basex-xqj-1.2.3.zip`. As an alternative, you can also find the libraries in the BaseX installation directory, in the `lib` sub-directory.

**BaseX XQJ Connection**

The next step is to create an XQJ connection as described in *How to Configure an XQJ Connection* on page 377.

For a default BaseX configuration, the following connection details apply (you can modify them when necessary):

- **Port**: 1984
- **serverName**: localhost
- **user**: admin
- **password**: admin

**XQuery Execution**

Now that the XQJ connection is configured, open the XQuery file you wish to execute in Oxygen XML Developer plugin and create a *Transformation Scenario* as described in *XQuery Transformation* on page 331. In the *Transformer* drop-down list, select the name of the XQJ connection you created. Apply the transformation scenario and the XQuery will be executed.
## Chapter 13

### Importing Data

This chapter describes how you can import data stored in text format, Excel sheet, or relational database tables, into XML documents.

**Topics:**

- *Introduction*
- *Import from Database*
- *Import from MS Excel Files*
- *Import from HTML Files*
- *Import from Text Files*
- *Import Content Dynamically*
Introduction

Computer systems and databases contain data in incompatible formats and one of the most time-consuming activities has been to exchange data between these systems. Converting the data to XML can greatly reduce complexity and create data that can be read by different types of applications.

This is why Oxygen XML Developer plugin offers support for importing text files, MS Excel files, Database Data, and HTML files into XML documents. The XML documents can be further converted into other formats using the Transform features.

Import from Database

This section explains how to import data from a database into Oxygen XML Developer plugin.

Import Table Content as XML Document

The steps for importing the data from a relational database table are the following:

1. Go to menu File > Import > oXygen / Database Data.

   Clicking this action opens a dialog box with all the defined database connections:
2. Select the connection to the database that contains the data.
   Only connections configured in relational data sources can be used to import data.
3. If you want to edit, delete, or add a data source or connection, click on the Configure Database Sources button.
   The Preferences/Data Sources option page is opened.
4. Click Connect.
5. From the catalogs list, click on a schema and choose the required table.
6. Click the OK button.

   The Import Criteria dialog box opens with a default query string in the SQL Query pane:
The dialog box contains the following options:

- **SQL Preview** - If the SQL Preview button is pressed, the Import settings pane displays the labels that are used in the XML document and the first five lines from the database. By default, all data items in the input are converted to element content, but this can be overridden by clicking the individual column headers. Clicking once on a column header causes the data from this column to be used as attribute values of the row elements. Click a second time and the data from that column is ignored when generating the XML file. You can cycle through these options by continuing to click the column header. The following symbols are used in the column header to indicate the type of content the column is converted to:
  - <> - data columns converted to element content
  - = - data columns converted to attribute content
  - x - ignored data

- **Change labels** - This button opens a new dialog box that allows you to edit the names of the root and row elements, change the XML name, and change the conversion criterion. The XML names can be edited by double-clicking the desired item and entering the required label. The conversion criterion can also be modified by selecting ELEMENT, ATTRIBUTE, or SKIPPED from the drop-down list.

- **Generate XML Schema** - Allows you to specify the path of the generated XML Schema file.
7. Click the *SQL Preview* button.

The *SQL Query* string is editable. You can specify which fields are considered.

Use aliases if the following statements are true:

- the query string represents a join operation of two or more tables
- columns that are selected from different tables have the same name

The use of aliases avoids the confusion of two columns being mapped to the same name in the result document of the importing operation.

```sql
select s.subcat_id,
    s.nr as s_nr,
    s.name,
    q.q_id,
    q.nr as q_nr,
    q.q_text
from faqsubcategory s,
    faq.question q
where ...
```

The input data is displayed in tabular form in the *Import settings* pane. The *XML Import Preview* pane contains an example of what the generated XML looks like.

**Convert Table Structure to XML Schema**

The structure of a table from a relational database can be imported in Oxygen XML Developer plugin as an XML Schema. This feature is activated by the *Generate XML Schema* option from the *Import criteria* dialog box used in the procedure for importing table data as an XML instance document.

**Import from MS Excel Files**

Oxygen XML Developer plugin offers support for importing MS Excel Files. To import Excel files, go to *File > Import > MS Excel file* and in the *Import* dialog box select the file you want to import. In the *Available Sheets* section of this dialog box, the sheets of the document you are importing are presented. Select a sheet and click next to move on to the second *Import* dialog box.
The Settings section presents the data from the Excel sheet in a tabular form. It also contains the following options:

- **First row contains field names** - Uses the content from the first row to name the columns.
- **Import formatted data (as displayed in Excel)** - Keeps the Excel styling.
- **Change labels** - Opens the Presentation Names dialog box. The following options are available:
  - **Root Element** - allows you to edit the name of the Root element.
  - **Row Element** - allows you to edit the name of the Row element.
  - **Real Name** - contains the original name of each Heading.
  - **XML Name** - allows you to modify the names of the Headings.
  - **Criterion** - allows you to transform the Heading elements to attributes of the Root element.
- **Import settings** - Opens the XML / Import preferences page.

The XML Import Preview section displays the Excel document in an XML format.

The Output File section contains the following options:

- **Open in Editor** - Opens the imported document in the Editor.
- **Save in File** - Saves the imported document in the specified location.

When you finish configuring the options in these dialog boxes, click **Import**.
Import from MS Excel 2007-2010 (.xlsx)

To import XML from Excel 2007-2010 (.xlsx) documents, Oxygen XML Developer plugin needs additional libraries from the release 3.10 of Apache POI project. Follow these steps:

2. From the downloaded project locate and add the following .jar files in the plugin.xml file:
   - dom4j-1.6.1.jar
   - poi-ooxml-3.10-FINAL-20140208.jar
   - poi-ooxml-schemas-3.10-FINAL-20140208.jar
   - xmlbeans-2.3.0.jar

Import from HTML Files

HTML is one of the formats that can be imported as an XML document. The steps needed are:

1. Go to menu File > Import > oXygen > HTML File ... The Import HTML wizard is displayed.
2. Enter the URL of the HTML document.
3. Select the type of the result XHTML document:
   - XHTML 1.0 Transitional
   - XHTML 1.0 Strict
4. Click the OK button.

The resulting document is an XHTML file containing a DOCTYPE declaration that references the XHTML DTD definition on the Web. The parsed content of the imported file is transformed to XHTML Transitional or XHTML Strict depending on what radio button you chose when performing the import operation.

Import from Text Files

The steps for importing a text file into an XML file are the following:

1. Go to menu File > Import > oXygen > Text File... The Select text file dialog box is displayed.
2. Select the URL of the text file.
3. Select the encoding of the text file.
4. Click the OK button.

The Import Criteria dialog box is displayed:

The input data is displayed in a tabular form. The XML Import Preview panel contains an example of what the generated XML document looks like. The names of the XML elements and the transformation of the first five lines from the text file are displayed in the Import settings section. All data items in the input are converted by default to element content, but this can be overridden by clicking the individual column headers. Clicking once a column header causes the data from this column to be used as attribute values of the row elements. Click the second time and the column's data is ignored when generating the XML file. You can cycle through these three options by continuing to click the column header. The following symbols decorate the column header to indicate the type of content that column is converted to:

- <> symbols for data columns converted to element content
- = symbol for data columns converted to attribute content
5. Select the field delimiter for the import settings:
   • Comma;
   • Semicolon;
   • Tab;
   • Space;
   • Pipe.

6. Set other optional settings of the conversion.

The dialog box offers the following settings:

• **First row contains field names** - If the option is enabled, you will notice that the table has moved up. The default column headers are replaced (where such information is available) by the content of the first row. In other words, the first row is interpreted as containing the field names. The changes are also visible in the preview of the XML document. To return to default settings (where the first row is interpreted as containing data and not fields names), simply uncheck the option.

• **Change labels** - This button opens a new dialog box that allows you to edit the names of the root and row elements, change the XML name and the conversion criterion.

  The XML names can be edited by double-clicking the desired item and entering the required label. The conversion criterion can also be modified by selecting one of the drop-down list options: **ELEMENT**, **ATTRIBUTE**, or **SKIPPED**.

• **Output file** - Allows you to select the output XML file.

---

**Import Content Dynamically**

Along with the built-in support for various useful URL protocols (such as HTTP or FTP), Oxygen XML Developer plugin also provides special support for a *convert* protocol that can be used to chain predefined processors to import content from various sources dynamically.

A dynamic conversion URL chains various processors that can be applied in sequence on a target resource and has the following general syntax:

```
convert:/processor=xslt;ss=urn:processors:excel2d.xsl/processor=excel!/urn:files:sample.xls
```

The previous example first applies a processor called `excel` on a target identified by the identifier `urn:files:sample.xls` and converts the Excel™ resource to XML. The second applied processor (`xslt`) applies an XSLT stylesheet identified using the identifier `urn:processors:excel2d.xsl` over the content resulting from the first applied processor. These identifiers are all mapped to real resources on disk via an *XML catalog* that is configured in the application, as in the following example:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteURI uriStartString="urn:files:" rewritePrefix="/resources/"/>
  <rewriteURI uriStartString="urn:processors:" rewritePrefix="/processors/"/>
</catalog>
```

This type of URL can be opened in the application by using the **Open URL...** action from the **File** menu. It can also be referenced from existing XML resources via **xi:include** or from **DITA Maps** as topic references.

A GitHub project that contains various dynamic conversion samples for producing DITA content from various sources (and then publishing it) can be found here: [https://github.com/oxygenxml/dita-glass](https://github.com/oxygenxml/dita-glass).
Conversion Processors

A set of predefined conversion processors is provided in Oxygen XML Developer plugin out-of-the-box. Each processor has its own parameters that can be set to control the behavior of the conversion process. All parameters that are resolved to resources are passed through the XML catalog mapping.

The following predefined conversion processors are included:

- **xslt Processor** - Converts an XML input using XSLT 2.0 processing. The `ss` parameter indicates the stylesheet resource to be loaded. All other specified parameters will be set as parameters to the XSLT transformation.
  
  ```
  convert:/processor=xslt;ss=urn:processors:convert.xsl;pl=v1!/urn:files:sample.xml
  ```

- **xquery Processor** - Converts an XML input using XQuery processing. The `ss` parameter indicates the XQuery script to be loaded. All other specified parameters will be set as parameters to the XSLT transformation.
  
  ```
  convert:/processor=xquery;ss=urn:processors:convert.xquery;pl=v1!/urn:files:sample.xml
  ```

- **excel Processor** - Converts an Excel™ input to an XML format that can be later converted by other piped processors. It has a single parameter `sn`, which indicates the name of the sheet that needs to be converted. If this parameter is missing, the XML will contain the combined content of all sheets included in the Excel™ document.
  
  ```
  convert:/processor=excel;sn=test!/urn:files:sample.xls
  ```

- **java Processor** - Converts an input to another format by applying a specific Java method. The `jars` parameter is a comma separated list of JAR libraries or folders, from which libraries will be loaded. The `ccn` parameter is the fully qualified name of the conversion class that will be instantiated. The conversion class needs to have a method with the following signature:
  
  ```
  public void convert(String systemID, String originalSourceSystemID, InputStream is, OutputStream os, LinkedHashMap<String, String> properties) throws IOException
  ```

  ```
  convert:/processor=java;jars=libs;ccn=test.JavaToXML!/urn:files:java/WSEditorBase.java
  ```

- **js Processor** - Converts an input to another format by applying a JavaScript method. The `js` parameter indicates the script that will be used. The `fn` parameter is the name of the method that will be called from the script. The method must take a string as an argument and return a string. If any of the parameters are missing, an error is thrown and the conversion stops.

  ```
  convert:/processor=js;js=urn:processors:md.js;fn=convertExternal!/urn:files:sample.md
  ```

- **json Processor** - Converts a JSON input to XML. It has no parameters.

  ```
  convert:/processor=json!/urn:files:personal.json
  ```

- **xhtml Processor** - Converts HTML content to well-formed XHTML. It has no parameters.

  ```
  convert:/processor=xhtml!/urn:files:test.html
  ```

- **wrap Processor** - Wraps content in a tag name making it well-formed XML. The `rn` parameter indicates the name of the root tag to use. By default, it is `wrapper`. The `encoding` parameter specifies the encoding that should be used to read the content. By default, it is UTF8. As an example, this processor can be used if you want to process a comma-separated values file with an XSLT stylesheet to produce XML content. The CSV file is first wrapped as well-formed XML, which is then processed with an `xslt` processor.

  ```
  convert:/processor=wrap!/urn:files:test.csv
  ```
Reverse Conversion Processors

All processors defined above can also be used for saving content back to the target resource if they are defined in the URL as reverse processors. Reverse processors are evaluated right to left. These reverse processors allow round-tripping content to and from the target resource.

As an example, the following URL converts HTML to DITA when the URL is opened using the h2d.xsl stylesheet and converts DITA to HTML when the content is saved in the application using the d2h.xsl stylesheet.

```xml
convert:/processor=xslt;ss=h2d.xsl/rprocessor=xslt;ss=d2h.xsl!/urn:files:sample.html
```
This chapter explains how you can integrate Oxygen XML Developer plugin with a content management system (CMS), to edit the data stored in the CMS directly in Oxygen XML Developer plugin.
Integration with Documentum (CMS) (deprecated)

Important: Starting with version 17.0, the support for Documentum (CMS) is deprecated and will no longer be actively maintained.

Oxygen XML Developer plugin provides support for browsing and managing Documentum repositories in the Data Source Explorer. You can easily create new resources on the repository, copy and move them using contextual actions or the drag and drop support, edit and transform the documents in the editor. The operations that can be performed on repository resources are described in the Documentum (CMS) actions section.

Oxygen XML Developer plugin supports Documentum (CMS) version 6.5 and 6.6 with Documentum Foundation Services 6.5 or 6.6 installed.

Attention:
It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

Configure Connection to Documentum Server

This section explains how to configure a connection to a Documentum server.

How to Configure a Documentum (CMS) Data Source

Available in the Enterprise edition only.

To configure a Documentum (CMS) data source you need the Documentum Foundation Services Software Development Kit (DFS SDK) corresponding to your server version. The DFS SDK can be found in the Documentum (CMS) server installation kit or it can be downloaded from EMC Community Network.

Note: The DFS SDK can be found in the form of an archive named, for example, emc-dfs-sdk-6.5.zip for Documentum (CMS) 6.5.

1. Open the Preferences dialog box and go to Data Sources.
2. In the Data Sources panel click the New button.
3. Enter a unique name for the data source.
4. Select Documentum (CMS) from the driver type combo box.
5. Press the Choose DFS SDK Folder button.
6. Select the folder where you have unpacked the DFS SDK archive file.

   If you have indicated the correct folder the following Java libraries (jar files) will be added to the list (some variation of the library names is possible in future versions of the DFS SDK):

   • lib/java/emc-bpm-services-remote.jar
   • lib/java/emc-ci-services-remote.jar
   • lib/java/emc-collaboration-services-remote.jar
   • lib/java/emc-dfs-rt-remote.jar
   • lib/java/emc-dfs-services-remote.jar
   • lib/java/emc-dfs-tools.jar
   • lib/java/emc-search-services-remote.jar
   • lib/java/ucf/client/ucf-installer.jar
   • lib/java/commons/**.jar (multiple jar files)
   • lib/java/jaxws/**.jar (multiple jar files)
   • lib/java/utils/**.jar (multiple jar files)

   Note: If for some reason the jar files are not found, you can add them manually by using the Add Files and Add Recursively buttons and navigating to the lib/java folder from the DFS SDK.
7. Click the **OK** button to finish the data source configuration.

**How to Configure a Documentum (CMS) Connection**

Available in the Enterprise edition only.

The steps for configuring a connection to a Documentum (CMS) server are the following:

1. **Open the Preferences dialog box** and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the previously configured Documentum (CMS) data sources in the **Data Source** combo box.
5. Fill-in the connection details:
   - **URL** - The URL to the Documentum (CMS) server: http://<hostname>:<port>
   - **User** - The user name to access the Documentum (CMS) repository.
   - **Password** - The password to access the Documentum (CMS) repository.
   - **Repository** - The name of the repository to log into.
6. Click the **OK** button to finish the configuration of the connection.

**Known Issues**

The following are known issues with the Documentum (CMS):

1. Please note that there is a known problem in the UCF Client implementation for Mac OS X from Documentum 6.5 which prevents you from viewing or editing XML documents from the repository on Mac OS X. The UCF Client is the component responsible for file transfer between the repository and the local machine. This component is deployed automatically from the server. Documentum 6.6 does not exhibit this problem.

   **Note:** This issue was reproduced with Documentum 6.5 SP1. In Documentum 6.6 this is no longer reproducing.

2. In order for the Documentum driver to work faster on Linux, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen XML Developer plugin startup scripts (or in the *.vmoptions file) the system property:

   -Djava.security.egd=file:/dev/.urandom

**Documentum (CMS) Actions in the Data Source Explorer View**

Oxygen XML Developer plugin allows you to browse the structure of a Documentum repository in the **Data Source Explorer** view and perform various operations on the repository resources.

You can drag and drop folders and resources to other folders to perform move or copy operations with ease. If the drag and drop is between resources (drag the child item to the parent item) you can create a relationship between the respective resources.
Figure 232: Browsing a Documentum repository

Actions Available on Connection

The contextual menu of a Documentum (CMS) connection in the Data Source Explorer view offers the following actions:

Configure Database Sources

Opens the Data Sources preferences page where you can configure both data sources and connections.

New Cabinet

Creates a new cabinet in the repository. The cabinet properties are:

- **Type** - The type of the new cabinet (default is `dm_cabinet`).
- **Name** - The name of the new cabinet.
- **Title** - The title property of the cabinet.
- **Subject** - The subject property of the cabinet.

Refresh

Refreshes the connection.

Actions Available on Cabinets / Folders

The actions available on a Documentum (CMS) cabinet in the Data Source Explorer view are the following:

New Folder

Creates a new folder in the current cabinet / folder. The folder properties are the following:

- **Path** - Shows the path where the new folder will be created.
- **Type** - The type of the new folder (default is `dm_folder`).
- **Name** - The name of the new folder.
- **Title** - The title property of the folder.
- **Subject** - The subject property of the folder.
New Document

Creates a new document in the current cabinet / folder. The document properties are the following:

- **Path** - Shows the path where the new document will be created.
- **Name** - The name of the new document.
- **Type** - The type of the new document (default is `dm_document`).
- **Format** - The document content type format.

Import

Imports local files / folders in the selected cabinet / folder of the repository. Actions available when performing an import:

- **Add Files** - Opens a file browse dialog box and allows you to select files to add to the list.
- **Add Folders** - Opens a folder browse dialog box that allows you to select folders to add to the list. The subfolders will be added recursively.
- **Edit** - Opens a dialog box where you can change the properties of the selected file / folder from the list.
- **Remove** - Removes the selected files / folders from the list.

Rename

Changes the name of the selected cabinet / folder.

Copy

Copies the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop while holding the `[Ctrl]` (`Meta` on Mac OS) key pressed.

Move

Moves the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop.

Delete

Deletes the selected cabinet / folder from the repository. The following options are available:

- **Folder(s)** - Allows you to delete only the selected folder or to delete recursively the folder and all subfolders and objects.
- **Version(s)** - Allows you to specify what versions of the resources will be deleted.
- **Virtual document(s)** - Here you can specify what happens when virtual documents are encountered. They can be either deleted either by themselves or together with their descendants.

Refresh

Performs a refresh of the selected node's sub-tree.

Properties

Displays the list of properties of the selected cabinet / folder.

Actions Available on Resources

The actions available on a Documentum (CMS) resource in the **Data Source Explorer** view are the following:

Edit

Checks out (if not already checked out) and opens the selected resource in the editor.

Edit with

Checks out (if not already checked out) and opens the selected resource in the specified editor / tool.

Open (Read-only)

Opens the selected resource in the editor.

Open with

Opens the selected resource in the specified editor / tool.
Check Out

Checks out the selected resource from the repository. The action is not available if the resource is already checked out.

Check In

Checks in the selected resource (commits changes) into the repository. The action is only available if the resource is checked out.

![Check In Dialog Box](image)

*Figure 233: Check In Dialog Box*

The following resource properties are available:

- **Name** - The resource name in the repository.
- **Version** - Allows you to choose what version the resource will have after being checked in.
- **Version label** - The label of the updated version.
- **Description** - An optional description of the resource.
- **Keep Locks** - When this option is enabled, the updated resource is checked into the repository but it also keeps it locked.
- **Make this the current version** - Makes the updated resource the current version (will have the CURRENT version label).

Cancel Checkout

Cancels the checkout process and loses all modifications since the checkout. Action is only available if the resource is checked out.

Export

Allows you to export the resource and save it locally.

Rename

Changes the name of the selected resource.

Copy

Copies the selected resource in a different location in the tree. Action is not available on virtual document descendants. This action can also be performed with drag and drop while holding the **Ctrl (Meta on OS X)** key pressed.

Move

Moves the selected resource in a different location in the tree. Action is not available on virtual document descendants and on checked out resources. This action can also be performed with drag and drop.
Delete
   Deletes the selected resource from the repository. Action is not available on virtual document descendants and on checked out resources.

Add Relationship
   Adds a new relationship for the selected resource. This action can also be performed with drag and drop between resources.

Convert to Virtual Document
   Allows you to convert a simple document to a virtual document. Action is available only if the resource is a simple document.

Convert to Simple Document
   Allows you to convert a virtual document to a simple document. Action is available only if the resource is a virtual document with no descendants.

Copy location
   Allows you to copy to clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.

Refresh
   Performs a refresh of the selected resource.

Properties
   Displays the list of properties of the selected resource.

Transformations on DITA Content from Documentum (CMS)
   Oxygen XML Developer plugin comes with the DITA Open Toolkit which is able to transform a DITA map to various output formats. However DITA Open Toolkit requires local DITA files so first you need to check out a local version of your DITA content.

Integration with Microsoft SharePoint
   This section explains how to work with a SharePoint connection in the Data Source Explorer view.

   Note: The SharePoint connection is available in the Enterprise edition.

   Note: You can access documents stored on SharePoint Online for Office 365.

To watch our video demonstration about connecting to a repository located on a SharePoint server and using SharePoint, go to http://www.oxygenxml.com/demo/SharePoint_Support.html and SharePoint Online for Office 365

How to Configure a SharePoint Connection
   By default Oxygen XML Developer plugin contains a predefined SharePoint data source. Use this data source to create a connection to a SharePoint server which will be available in the Data Source Explorer view.

   Follow these steps to configure a SharePoint connection:

   1. Open the Preferences dialog box and go to Data Sources.
   2. In the Connections panel click the New button.
   3. Enter a unique name for the connection.
   4. Select SharePoint in the Data Source combo box.
   5. Fill-in the connection details:
      a) Set the URL to the SharePoint repository in the field SharePoint URL.
      b) Set the server domain in the Domain field.
c) Set the user name to access the SharePoint repository in the User field.
d) Set the password to access the SharePoint repository in the Password field.
To watch our video demonstration about connecting to repository located on a SharePoint server, go to http://www.oxygenxml.com/demo/SharePoint_Support.html.

SharePoint Connection Actions

This section explains the actions that are available on a SharePoint connection in the Data Source Explorer view.

**Actions Available at Connection Level**
The contextual menu of a SharePoint connection in the Data Source Explorer view contains the following actions:

- **Configure Database Sources...**
  Opens the Data Sources preferences page. Here you can configure both data sources and connections.

- **Disconnect**
  Stops the connection.

- **New Folder...**
  Creates a new folder on the server.

- **Import Files...**
  Allows you to add a new file on the server.

- **Refresh**
  Performs a refresh of the connection.

- **Find/Replace in Files...**
  Allows you to find and replace text in multiple files from the server.

**Actions Available at Folder Level**
The contextual menu of a folder node in a SharePoint connection in the Data Source Explorer view contains the following actions:

- **New File**
  Creates a new file on the server in the current folder.

- **New Folder...**
  Creates a new folder on the server.

- **Import Folders...**
  Imports folders on the server.

- **Import Files**
  Allows you to add a new file on the server in the current folder.

- **Cut**
  Removes the current selection and places it in the clipboard.

- **Copy**
  Copies the current selection.

- **Paste**
  Pastes the copied selection.

- **Rename**
  Allows you to change the name of the selected folder.

- **Delete**
  Removes the selected folder.
**Actions Available at File Level**

The contextual menu of a file node in a SharePoint connection in the **Data Source Explorer** view contains the following actions:

- **Open**
  Allows you to open the selected file in the editor.

- **Cut**
  Removes the current selection and places it in the clipboard.

- **Copy**
  Copies the current selection into the clipboard.

- **Copy Location**
  Copies an application specific URL for the selected resource to the clipboard. You can use this URL for various actions like opening or transforming the resources.

- **Check Out**
  Checks out the selected document on the server.

- **Check In**
  Checks in the selected document on the server. This action opens the **Check In** dialog. In this dialog, the following options are available:
    - **Minor Version** - increments the minor version of the file on the server
    - **Major Version** - increments the major version of the file on the server
    - **Overwrite** - overwrites the latest version of the file on the server
    - **Comment** - allows you to comment on a file that you check in

- **Discard Check Out**
  Discards the previous checkout operation, making the file available for editing to other users.

- **Rename**
  Allows you to change the name of the selected file.

- **Delete**
  Removes the selected file.

- **Refresh**
  Performs a refresh of the selected node.

- **Properties**
  Displays the properties of the current file in a **Properties** dialog box.

- **Find/Replace in Files...**
  Allows you to find and replace text in multiple files from the server.

**Note:** The **Check In**, **Check Out**, and **Discard Check Out** options are available in the Enterprise edition only.
Chapter 15

Tools

Topics:

- XML Digital Signatures

Oxygen XML Developer plugin ships with tools to support XML digital signatures.
XML Digital Signatures

This chapter explains how to apply and verify digital signatures on XML documents.

Overview

Digital signatures are widely used as security tokens, not just in XML. A digital signature provides a mechanism for assuring integrity of data, the authentication of its signer, and the non-repudiation of the entire signature to an external party:

- A digital signature must provide a way to verify that the data has not been modified or replaced to ensure integrity.
- The signature must provide a way to establish the identity of the data's signer for authentication.
- The signature must provide the ability for the data's integrity and authentication to be provable to a third party for non-repudiation.

A public key system is used to create the digital signature and it's also used for verification. The signature binds the signer to the document because digitally signing a document requires the originator to create a hash of the message and then encrypt that hash value with his own private key. Only the originator has that private key and he is the only one that can encrypt the hash so that it can be unencrypted using his public key. The recipient, upon receiving both the message and the encrypted hash value, can decrypt the hash value, knowing the originator's public key. The recipient must also try to generate the hash value of the message and compare the newly generated hash value with the unencrypted hash value received from the originator. If the hash values are identical, it proves that the originator created the message, because only the actual originator could encrypt the hash value correctly.

XML Signatures can be applied to any digital content (data object), including XML (see W3C Recommendation, *XML-Signature Syntax and Processing*). An XML Signature may be applied to the content of one or more resources:

- enveloped or enveloping signatures are applied over data within the same XML document as the signature
- detached signatures are applied over data external to the signature element; the signature is "detached" from the content it signs. This definition typically applies to separate data objects, but it also includes the instance where the signature and data object reside within the same XML document but are sibling elements.

The XML Signature is a method of associating a key with referenced data. It does not normatively specify how keys are associated with persons or institutions, nor the meaning of the data being referenced and signed.

The original data is not actually signed. Instead, the signature is applied to the output of a chain of canonicalization and transformation algorithms, which are applied to the data in a designated sequence. This system provides the flexibility to accommodate whatever "normalization" or desired preprocessing of the data that might be required or desired before subjecting it to being signed.

To canonicalize something means to put it in a standard format that everyone generally uses. Because the signature is dependent on the content it is signing, a signature produced from a not canonicalized document could possibly be different from one produced from a canonicalized document. The canonical form of an XML document is physical representation of the document produced by the method described in this specification. The term canonical XML refers to XML that is in canonical form. The XML canonicalization method is the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term XML canonicalization refers to the process of applying the XML canonicalization method to an XML document or document subset. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would 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 XML Developer plugin: Canonical XML (or Inclusive XML Canonicalization) (XMLC14N) and Exclusive XML Canonicalization (EXCC14N). The first is used for XML where the context doesn’t change while the second was designed for canonicalization where the context might change.

Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature. In this way all the declarations you might use will be unambiguously specified. A problem appears when the signed XML is moved into another XML document which has other declarations because the Inclusive Canonicalization will copy 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.

The three operations. Canonicalize..., Sign..., and Verify Signature..., are available from the Source submenu when invoking the contextual menu in Text mode or from the XML Tools menu.

Canonicalizing Files

You can select the canonicalization algorithm to be used for a document from the dialog box that is displayed by using the Canonicalize action that is available from the Source submenu when invoking the contextual menu in Text mode or from the XML Tools menu.

Figure 234: Canonicalization Settings Dialog Box

You can set the following:

- **URL** - Specifies the location of the input URL.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
• **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
• **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
• **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
• **XPath** - The XPath expression provides the fragments of the XML document to be signed.
• **Output** - Specifies the output file path where the signed XML document will be saved.
• **Open in editor** - If checked, the output file will be opened in the editor.

**Certificates**

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called **keystores**.

A **keystore** is an encrypted file that contains private keys and certificates. All **keystore** entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No **keystore** can store an entity if its alias already exists in that **keystore** and cannot store trusted certificates generated with keys in its **keystore**.

In Oxygen XML Developer plugin there are provided two types of **keystores**: Java Key Store (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A **keystore** file is protected by a password. In a PKCS 12 **keystore** you should not store a certificate without alias together with other certificates, with or without alias, as in such a case the certificate without alias cannot be extracted from the **keystore**.

To configure the options for a certificate or to validate it, *open the Preferences dialog box* and go to **Certificates**.

**Signing Files**

You can select the type of signature to be used for documents from a signature settings dialog. To open this dialog, select the **Sign...** action from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu.
The following options are available:

- **Input** - Specifies the location of the input URL.
- **Transformation Options:**
  - **None** - If selected, no canonicalization algorithm is used.
  - **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
  - **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
  - **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
  - **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **ID** - Provides ID of the XML element to be signed.
- **Envelope** - If selected, the *enveloped* signature is used.
- **Detached** - If selected, the *detached* signature is used.
- **Append KeyInfo** - If this option is checked, the `ds:KeyInfo` element will be added in the signed document.
- **Signature algorithm** - The algorithm used for signing the document. The following options are available: **RSA with SHA1**, **RSA with SHA256**, **RSA with SHA384**, and **RSA with SHA512**.
- **Output** - Specifies the path of the output file where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

**Note:** If Oxygen XML Developer plugin could not find a valid certificate, a link is provided at the top of the dialog that opens the *XML Signing Certificates preferences page* where you can configure a valid certificate.
Verifying the Signature

You can verify the signature of a file by selecting the **Verify Signature** action from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu. The **Verify Signature** dialog then allows you to specify the location of the file whose signature is verified.

If the signature is valid, a dialog displays the name of the signer. Otherwise, an error shows details about the problem.
The Oxygen XML Developer plugin includes a number of extension points, which can be implemented by other Eclipse plugins that depend on it. All of them are listed in the plugin.xml file, along with samples of usage code. The following is a list with short descriptions for some of the most useful extension points:

**Extension point: ditaKeyDefinitionManager**
It can be used to provide an external keys manager, responsible of providing DITA keys that are then used for editing and resolving referenced content. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/ditaKeyDefinitionManager.exsd.

**Extension point: actionBarContributorCustomizer**
A very useful extension point that can add or remove actions from various menus, contextual menus, and toolbars that are contributed by the Oxygen XML Developer plugin. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/actionBarContributorCustomizer.exsd.

**Extension point: customEditorInputCreator**
Create your custom editor input for a certain resource that will be opened by the Oxygen XML Developer plugin when clicking links. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/customEditorInputCreator.exsd.

**Extension point: editorAdapterContributor**
When an adapter is requested to the opened XML editor you can provide your custom adapter from your external plugin. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/editorAdapterContributor.exsd.

**Extension point: extensionsBundleContributor**
Provide your own ExtensionsBundle implementation for a certain opened XML resource. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/extensionsBundleContributor.exsd.

**Extension point: stylesFilterContributor**
Provide your own StylesFilter implementation for special visual rendering when an XML resource is opened in the Author editing mode. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/stylesFilterContributor.exsd.

**Extension point: XMLRefactoringContributor**
Contribute a folder that contains the additional XML Refactoring operation descriptor files and XQuery scripts that can be used by the batch XML refactoring actions. Its EXSD schema can be found in: OXYGEN_PLUGIN_DIR/exsd-schema/xmlRefactoringContributor.exsd.
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.

Topics:

- Preferences
- Importing / Exporting Global Options
- Reset Global Options
- Customizing Default Options
- Scenarios Management
- Editor Variables
- Localization of the User Interface
Preferences

You can configure Oxygen XML Developer plugin options using the Preferences dialog box.

To open the preferences dialog box, go to Window (Eclipse on Mac OSX) and choose Preferences > Oxygen XML Developer.

You can restore options to their default values by pressing the Restore Defaults button, available in each preferences page.

Press ? or F1 for help on any preferences page.

A limited version of the Preferences dialog box is available from the contextual menu in the editor.

Figure 236: Eclipse Preferences Dialog Box - Restricted Version

Global options and license information are stored in the following locations:

- [user-home-folder]\Application Data\com.oxygenxml.developer for Windows XP
- [user-home-folder]\AppData\Roaming\com.oxygenxml.developer for Windows Vista/7
- [user-home-folder]/Library/Preferences/com.oxygenxml.developer for Mac OS X
- [user-home-folder]/.com.oxygenxml.developer for Linux

Oxygen XML Developer plugin License

To configure the license options, open the Preferences dialog box. This preferences page presents the details of the license key that enables the Oxygen XML Developer plugin plugin, such as registration name, category and number of
purchased licenses, encrypted signature of the license key. Clicking the Register button opens the Oxygen XML Developer plugin License dialog box that allows you to insert a new license key.

**Archive Preferences**

To configure Archive preferences, open the Preferences dialog box and go to Archive.

The following options are available in the Archive preferences panel:

- **Archive backup options** - Controls if the application makes backup copies of the modified archives. The following options are available:
  - Always create backup copies of modified archives - When you modify an archive, its content is backed up.
  - Never create backup copies of modified archives - No backup copy is created.
  - Ask for each archive once per session - Once per application session for each modified archive, user confirmation is required to create the backup. This is the default setting.

  Note: Backup files have the name originalArchiveFileName.bak and are located in the same folder as the original archive.

- **Show archive backup dialog** - Select this option if you want to be notified for backup when modifying in archives. The last backup option you chose will always be used as the default one.

- **Archive types** - This table contains all known archive extensions mapped to known archive formats. Each row maps a list of extensions to an archive type supported in Oxygen XML Developer plugin. You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.

  Important: You have to restart Oxygen XML Developer plugin after removing an extension from the table in order for that extension to not be recognised anymore as an archive extension.

- **Store Unicode file names in Zip archives** - Use this option when you archive files that contain international (that is, non-English) characters in file names or file comments. If this option is selected and an archive is modified in any way, UTF-8 characters are used in the names of all files in the archive.

**CSS Validator Preferences**

To configure the CSS Validator preferences, open the Preferences dialog box and go to CSS Validator.

You can configure the following options for the built-in CSS Validator of Oxygen XML Developer plugin:

- **Profile** - Selects one of the available validation profiles: CSS 1, CSS 2, CSS 2.1, CSS 3, CSS 3 with Oxygen extensions, SVG, SVG Basic, SVG Tiny, Mobile, TV Profile, ATSC TV Profile. The CSS 3 with Oxygen extensions profile includes all the CSS 3 standard properties and the CSS extensions specific for Oxygen. That means all Oxygen specific extensions are accepted in a CSS stylesheet by the built-in CSS validator when this profile is selected.

- **Media type** - Selects one of the available mediums: all, aural, braille, embossed, handheld, print, projection, screen, tty, tv, presentation, oxygen

- **Warning level** - Sets the minimum severity level for reported validation warnings. Can be one of: All, Normal, Most Important, No Warnings

- **Ignore properties** - You can type comma separated patterns that match the names of CSS properties that will be ignored at validation. As wildcards you can use:
  - * to match any string
  - ? to match any character

- **Recognize browser CSS extensions (applies also to content completion)** - If checked, Oxygen XML Developer plugin recognizes (no validation is performed) browser-specific CSS properties. The Content Completion Assistant lists these properties at the end of its list, prefixed with the following particles:
  - -moz- for Mozilla
  - -ms- for Internet Explorer
Custom Editor Variables Preferences

An editor variable is useful for making a transformation scenario, a validation scenario or an external tool independent of the file path on which the scenario / command line is applied. An editor variable is specified as a parameter in a transformation scenario, validation scenario or command line of an external tool. Such a variable is defined by a name, a string value and a text description. A custom editor variable is defined by the user and can be used in the same expressions as the built-in ones.

![Custom Editor Variables](image)

Figure 237: Custom Editor Variables

Data Sources Preferences

To configure the Data Sources preferences, open the Preferences dialog box and go to Data Sources.

Data Sources Preferences

To configure the Data Sources preferences, open the Preferences dialog box and go to Data Sources. In this preferences page you can configure data sources and connections to relational databases as well as native XML databases. You can check the list of drivers (http://www.oxygenxml.com/database_drivers.html) available for the major database servers.

![Data Sources](image)

Figure 238: The Data Sources Preferences Panel

- **New** - opens the Data Sources Drivers dialog that allows you to configure a new database driver.
Figure 239: The Data Sources Drivers Dialog

The following options are available:

- **Name** - The name of the new data source driver that will be used for creating connections to the database.
- **Type** - Selects the data source type from the supported driver types.
- **Help** - Opens the User Manual at the list of the sections where the configuration of supported data sources is explained and the URLs for downloading the database drivers are specified.
- **Driver Class** - Specifies the driver class for the data source driver.
- **Add** - Adds the driver class library.
- **Remove** - Removes the selected driver class library from the list.
- **Detect** - Detects driver class candidates.
- **Stop** - Stops the detection of the driver candidates.

- **Edit** - Opens the Data Sources Drivers dialog for editing the selected driver. See above the specifications for the Data Sources Drivers dialog. In order to edit a data source, there must be no connections using that data source driver.
- **Delete** - Deletes the selected driver. In order to delete a data source, there must be no connections using that data source driver.
Figure 240: The Connections Preferences Panel

For performance issues, you can set the maximum number of cells that will be displayed in the Table Explorer view for a database table. Leave the field Limit the number of cells empty if you want the entire content of the table to be displayed. By default this field is set to 2,000. If a table having more cells than the value set here is displayed in the Table Explorer view, a warning dialog will inform you that the table is only partially shown.

In Oracle XML a container can hold millions of resources. If the node corresponding to such a container in the Data Source Explorer view would display all the contained resources at the same time the performance of the view would be very slow. To prevent such a situation only a limited number of the contained resources is displayed as child nodes of the container node. Navigation to other contained resources from the same container is enabled by the Up and Down buttons in the Data Source Explorer view. This limited number is set in the option Maximum number of children for container nodes. The default value is 200 nodes.

The Show warning when expanding other database schema option controls if a warning message will be displayed when expanding another database schema and there are tables selected in the current expanded one. This applies for the dialog Select database table when invoking the Convert DB Structure to XML Schema action.

The actions of the buttons from the Connections panel are the following:

- **New** - opens the Connection dialog which has the following fields:
Figure 241: The Connection Dialog

- **Name** - The name of the new connection that will be used in transformation scenarios and validation scenarios.
- **Data Source** - Allows selecting a data source defined in the Data Source Drivers dialog.

Depending upon the selected data source, you can set some of the following parameters in the Connection details area:

- **URL** - The URL for connecting to the database server.
- **User** - The user name for connecting to the database server.
- **Password** - The password of the specified user name.
- **Host** - The host address of the server.
- **Port** - The port where the server accepts the connection.
- **XML DB URI** - The database URI.
- **Database** - The initial database name.
- **Collection** - One of the available collections for the specified data source.
- **Environment home directory** - Specifies the home directory (only for a Berkeley database).
- **Verbosity** - Sets the verbosity level for output messages (only for a Berkeley database).
- **Use a secure HTTPS connection (SSL)** - Allows you to establish a secure connection to an eXist database through the SSL protocol.

- **Edit** - Opens the Connection dialog, allowing you to edit the selected connection. See above the specifications for the Connection dialog.
- **Duplicate** - Creates a duplicate of the currently selected connection.
- **Delete** - Deletes the selected connection.

**Download Links for Database Drivers**

Below you can find instructions for getting the drivers that are necessary to access databases in Oxygen XML Developer plugin.

- **Berkeley DB XML database** - Copy the jar files from the Berkeley database install directory into the Oxygen XML Developer plugin install directory as described in the procedure for configuring a Berkeley DB data source.
- **IBM DB2 Pure XML database** - Go to the IBM website and in the DB2 Clients and Development Tools category select the DB2 Driver for JDBC and SQLJ download link. Fill out the download form and download the zip file.
Unzip the zip file and use the db2jcc.jar and db2jcc_license_cu.jar files in Oxygen XML Developer plugin for configuring a DB2 data source.

- **eXist database** - Copy the jar files from the eXist database install directory to the Oxygen XML Developer plugin install directory as described in the procedure for configuring an eXist data source.
- **MarkLogic database** - Download the MarkLogic driver from MarkLogic Community site.
- **Oracle 11g database** - Go to the Oracle website and download the Oracle 11g JDBC driver called ojdbc6.jar.
- **PostgreSQL 8.3 database** - Go to the PostgreSQL website and download the PostgreSQL 8.3 JDBC driver called postgresql-8.3-603.jdbc3.jar.
- **Documentum xDb (X-Hive/DB) 10 XML database** - Copy the jar files from the Documentum xDb (X-Hive/DB) 10 database install directory to the Oxygen XML Developer plugin install directory as described in the procedure for configuring a Documentum xDb (X-Hive/DB) 10 data source.

**Table Filters Preferences**

to configure the Table Filters preferences, open the Preferences dialog box and go to Data Sources > Table Filters. Here you can choose which of the database table types will be displayed in the Data Source Explorer view.

![Table Filters Preferences Panel](image)

**DITA Preferences**

To access the DITA Preferences page, open the Preferences dialog box and click on DITA.

The **DITA-OT directory** option specifies the directory of the DITA Open Toolkit distribution to be used, by default, for validating and publishing DITA content. You can either provide a new file path for the specific DITA OT that you want to use or you can select a previously used one from the drop-down list.

**Note:** The DITA Open Toolkit is bundled with the Oxygen XML Developer plugin installation.

The **Show console output** option allows you to specify when to display the console output log. The following options are available:

- **When build fails** - displays the console output log if the build fails.
- **Always** - displays the console output log, regardless of whether or not the build fails.

**Document Type Association Preferences**

Oxygen XML Developer plugin uses document type associations to associate a document type with a set of functionality provided by a framework. To configure the Document Type Association options, open the Preferences dialog box and go to Document Type Association.

The following actions are available in the preferences panel:

- **Discover more frameworks by using add-ons update sites** - specifies update site URLs for framework add-ons
• **Document types table** - presents the currently defined document type associations, ordered by priority and alphabetically. Each edited document type has a set of association rules (used by the application to detect the proper document type association to use for an opened XML document). A rule is described by:

  • **Namespace** - specifies the namespace of the root element from the association rules set (*any* by default). If you want to apply the rule only when the root element has no namespace, leave this field empty (remove the `ANY_VALUE` string)
  • **Root local name** - specifies the local name of the root element (*any* by default)
  • **File name** - specifies the name of the file (*any* by default)
  • **Public ID** - represents the Public ID of the matched document
  • **Java class** - presents the name of the Java class, which is used to determine if a document matches the rule

• **New** - opens a dialog box that allows you to add a new association
• **Edit** - opens a new dialog that allows you to edit an existing association

  **Note:** If you try to edit an existing association type when you do not have write permissions to its store location, a dialog box will be shown, asking if you want to extend the document type.

• **Duplicate** - opens a new dialog that allows you to duplicate the configuration of an existing document type association
• **Extend** - extend an existing document type, allowing you to add or remove functionality, starting from a base document type. All of these changes will be saved as a patch. When the base document type is modified and evolves (for example, from one application version to another) the extension will evolve along with the base document type, allowing it to use the new actions added in the base document type.
• **Delete** - deletes the selected associations

• **Enable DTD/XML Schema processing in document type detection** - when this option is enabled, the matching process also examines the DTD/XML Schema associated with the document. For example, the fixed attributes declared in the DTD for the root element are also analyzed, if this is specified in the association rules. This is especially useful if you are writing DITA customizations. DITA topics and maps are also matched by looking for the `DITAArchVersion` attribute of the root element. This attribute is specified as default in the DTD and it is detected in the root element, helping Oxygen XML Developer plugin to correctly match the DITA customization.

  **(This option is enabled by default)**

• **Only for local DTD's / XML Schemas** - when the previous feature is enabled, you can choose with this option to process only the local DTD's / XML Schemas

  **(This option is enabled by default)**

• **Enable DTD/XML Schema caching** - when this option is enabled, the associated DTDs or XML Schema are cached when parsed for the first time, improving performance when opening new documents with similar schema associations

  **(This option is enabled by default)**

### Locations Preferences

Oxygen XML Developer plugin allows you to change the location where frameworks are stored, and to specify additional framework directories. The **Locations** preferences page allows you to specify the main frameworks folder location. You can choose between the **Default** directory (`[OXYGEN_DIR]/frameworks`) or a **Custom** specified directory. You can also change the current frameworks folder location value using the `com.oxygenxml.editor.frameworks.url` system property.

A list of additional frameworks directories can also be specified. The application will look in each of those folders for additional document type configurations to load. Use the **Add**, **Edit** and **Delete** buttons to manage the list of folders.

A document type (configuration) can be loaded from the following locations:

• **internal preferences** - The document type configuration is stored in the application's **Internal preferences**
• **additional frameworks directories** - The document type configuration is loaded from one of the specified **Additional frameworks directories** list
• **the frameworks folder** - The main folder containing framework configurations
All loaded document type configurations are first sorted by priority, then by document type name and then by load location (in the exact order specified above). When an XML document is opened, the application chooses the first document type configuration from the sorted list which matches the specific document.

All loaded document type configurations are first sorted by priority, then by document type

**The Document Type Dialog**

This dialog allows you to create or edit a *Document Type Association*. The following fields are available in this dialog:

- **Name** - the name of the *Document Type Association*
- **Storage** - displays the type of location where the framework configuration file is stored. Can be one of: **External** (framework configuration is saved in a file) or **Internal** (framework configuration is stored in the application's internal options)

  **Note:** If you set the Storage to Internal and the document type association settings are already stored in a framework file, the file content is saved in the application's internal options and the file is removed.

- **Description** - a detailed description of the framework
- **Priority** - depending on the priority level, Oxygen XML Developer plugin establishes the order in which the existing document type associations are evaluated to determine the type of a document you are opening. It can be one of the following: Lowest, Low, Normal, High, or Highest. You can set a higher priority to *Document Type Associations* you want to be evaluated first.
- **Initial edit mode** - sets the default edit mode when you open a document for the first time

You are able to configure the options of each *framework* in the following tabs:

- **Association rules**
- **Schema**
- **Classpath**
- **Author**
- **Templates**
- **Catalogs**
- **Transformation**
- **Validation**
- **Extensions**

**The Association Rules Tab**

By combining multiple association rules you can instruct Oxygen XML Developer plugin to identify the type of a document. An Oxygen XML Developer plugin association rule holds information about *Namespace, Root local name, File name, Public ID, Attribute*, and *Java class*. Oxygen XML Developer plugin identifies the type of a document when the document matches at least one of the *association rules*. Using the **Document type rule** dialog box, you can create *association rules* that activate on any document matching all the criteria in the dialog box.

In the **Association rules** tab you can perform the following actions:

- **New**
  Opens the **Document type rule** dialog box allowing you to create *association rules*.

- **Edit**
  Opens the **Document type rule** dialog box allowing you to edit the properties of the currently selected *association rule*.

- **Delete**
  Deletes the currently selected *association rules*.

- **Move Up**
  Moves the selection to the previous *association rule*. 
Move Down

Moves the selection to the following association rule.

The Schema Tab

In the Schema tab you can specify a schema that Oxygen XML Developer plugin uses in case an XML document does not contain a schema declaration and no default validation scenario is associated with it.

To set the Schema URL, use editor variables to specify the path to the Schema file.

Note: It is a good practice to store all resources in the framework directory and use the \${framework} editor variable to reference them. This is a recommended approach to designing a self-contained document type that can be easily maintained and shared between different users.

The Classpath Tab

The Classpath tab displays a list of folders and JAR libraries that hold implementations for API extensions, implementations for custom Author operations, different resources (such as stylesheets), and framework translation files. Oxygen XML Developer plugin loads the resources looking in the folders in the order they appear in the list.

In the Classpath tab you can perform the following actions:

+ New
  Opens a dialog box that allows you to add a resource in the Classpath tab.

Edit

Opens a dialog box that allows you to edit a resource in the Classpath tab.

Delete

Deletes the currently selected resource.

Move Up

Moves the selection to the previous resource.

Move Down

Moves the selection to the following resource.

The Author Tab

The Author tab is a container that holds information regarding the CSS file used to render a document in the Author mode, and regarding framework-specific actions, menus, contextual menus, toolbars, and content completion list of proposals.

The options that you configure in the Author tab are grouped in the following sub-tabs: CSS, Actions, Menu, Contextual menu, Toolbar, Content Completion.

CSS

The CSS sub-tab contains the CSS files that Oxygen XML Developer plugin uses to render a document in the Author mode. In this sub-tab, you can set alternate CSS files. When you are editing a document in the Author mode, you can switch between these CSS files from the Styles drop-down list on the Author Styles toolbar.

The following actions are available in the CSS sub-tab:

+ New
  Opens a dialog that allows you to add a CSS file.

Edit

Opens a dialog that allows you to edit a CSS file.

Delete

Deletes the currently selected CSS file.
Move Up
Moves the selection to the previous CSS file.

Move Down
Moves the selection to the following CSS file.

Enable multiple selection of alternate CSSs
Allows users to apply multiple alternate styles, as layers, over the main CSS style. This option is enabled by default for DITA document types.

ignore CSSs from the associated document type
The CSS files set in the CSS tab are overwritten by the CSS files specified in the document itself.

merge them with CSSs from the associated document type
The CSS files set in the CSS tab are merged with the CSS files specified in the document itself.

Actions
The Actions sub-tab holds the framework specific actions. Each action has an unique ID, a name, a description, and a shortcut key.

The following actions are available in this sub-tab:

New
Opens the Action dialog that allows you to add an action.

Duplicate
Duplicates the currently selected action.

Edit
Opens a dialog that allows you to edit an existing action.

Delete
Deletes the currently selected action.

The Action Dialog Box
To edit an existing document type action or create a new one, open the Preferences dialog box, go to Document Type Association, select a document type, and click Edit or New. The Document type dialog box is presented. In this dialog box, go to the Author tab, click Actions, select an action, and click Edit, or to create a new action click New.
Figure 243: The Action Dialog Box

The following options are available in the Action dialog box:

- **ID** - Specifies a unique action identifier.
- **Name** - Specifies the name of the action. This name is displayed as a tooltip or as a menu item.
- **Menu access key** - In Windows, the menu items are accessed using the Alt "Letter" shortcut when the menu is visible. The letter is visually represented by underlining the first occurrence of the letter in the menu item Name.
- **Description** - A description of the action.
- **Large icon** - Allows you to select an image for the icon that Oxygen XML Developer plugin uses for the toolbar action.

  **Tip:** A good practice is to store the image files inside the framework directory and use the $\{frameworks\}$ editor variable to make the image relative to the framework location. If the images are bundled in a jar archive (for instance, along with some Java operations implementation), it is convenient to reference the images by their relative path location in the class-path.

- **Small icon** - Allows you to select an image for the icon that Oxygen XML Developer plugin uses for the contextual menu action.
- **Shortcut key** - This field allows you to configure a shortcut key for the action that you are editing. The + character separates the keys. If the Enable platform-independent shortcut keys checkbox is enabled, the shortcut that you specify in this field is platform-independent and the following modifiers are used:
  - M1 represents the Command key on MacOS X, and the Ctrl key on other platforms.
  - M2 represents the Shift key.
• M3 represents the **Option** key on MacOS X, and the **Alt** key on other platforms.

• M4 represents the **Ctrl** key on MacOS X, and is undefined on other platforms.

### Operations

In this section of the **Action** dialog box, you configure the functionality of the action that you are editing. An action has one or more operation modes. The evaluation of an XPath expression activates an operation mode. The first enabled operation mode is activated when you trigger the action. The scope of the XPath expression must consist only of element nodes and attribute nodes of the edited document. Otherwise, the XPath expression does not return a match and does not fire the action. For more details see: **Action Mode Activation using XPath Expressions** on page 498.

The following options are available in this section:

- **When this XPath expression is true** - An XPath 2.0 expression that applies to elements and attributes. For more details see: **Action Mode Activation using XPath Expressions** on page 498.
- **invoke the operation** - Specifies the invoked operation.
- **with the arguments** - Specifies the arguments of the invoked operation.
- **Edit** - Allows you to edit the arguments of the operation.

- **Operation priority** - Increases or decreases the priority of an operation. The operations are invoked in the order of their priority. If more than one XPath expression is true, the operation with the highest priority is invoked.
  - **Add** - Adds an operation.
  - **Remove** - Removes an operation.
  - **Duplicate** - Duplicates an operation.

- **Evaluate activation XPath expressions even in read-only contexts** - If this checkbox is enabled, the action can be invoked even when the caret is placed in a read-only location.

### Action Mode Activation using XPath Expressions

An Author extension action can have multiple modes, each mode invoking an Author Operation with certain configured parameters. Each action mode has an XPath 2.0 expression for activating it.

For each action mode the application will check if the XPath expression is fulfilled (when it returns a not empty nodes set or a **true** result). If it is fulfilled, the operation defined in the action mode will be executed.

Two special XPath extension functions are provided: theoxy:allows-child-element() function that you can use to check whether an element is valid in the current context, considering the associated schema and theoxy:current-selected-element() function that you can use to get the currently selected element.

#### The oxy:allows-child-element() Function

This extension function allows author actions to be available in a context only if the associated schema permits it.

The **oxy:allows-child-element()** is evaluated at the caret position and has the following signature:

```xml
oxy:allows-child-element($childName, ($attributeName, $defaultAttributeValue, $contains?)?)
```

The following parameters are supported:

- **childName**

  the name of the element that you want to check whether it is valid in the current context. Its value is a string that supports the following forms:

  - the child element with the specified local name that belongs to the default namespace.

    ```xml
    oxy:allows-child-element("para")
    ```

  The above example verifies if the para element (of the default namespace) is allowed in the current context.

  - the child element with the local name specified by any namespace.

    ```xml
    oxy:allows-child-element("*:para")
    ```
The above example verifies if the `para` element (of any namespace) is allowed in the current context.

- a qualified name of an element.

```xml
oxy:allows-child-element("prefix:para")
```

The prefix is resolved in the context of the element where the caret is located. The function matches on the element with the `para` local name from the previous resolved namespace. In case the prefix is not resolved to a namespace, the function returns `false`.

- any element.

```xml
oxy:allows-child-element("*")
```

The above function verifies if any element is allowed in the current context.

Note: A common use case of `oxy:allows-child-element("*")` is in combination with the `attributeName` parameter.

**attributeName**

the attribute of an element that you want to check whether it is valid in the current context. Its value is a string that supports the following forms:

- the attribute with the specified name from no namespace.

```xml
oxy:allows-child-element("*", "class", " topic/topic ")
```

The above example verifies if an element with the `class` attribute and the default value of this attribute (that contains the `topic/topic` string) is allowed in the current context.

- the attribute with the local name specified by any namespace.

```xml
oxy:allows-child-element("*", "*:localname", " topic/topic ")
```

- a qualified name of an attribute.

```xml
oxy:allows-child-element("*", "prefix:localname", " topic/topic ")
```

The prefix is resolved in the context of the element where the caret is located. In case the prefix is not resolved to a namespace, the function returns `false`.

**defaultAttributeValue**

a string that represents the default value of the attribute. Depending on the value of the next parameter the default value of the attribute must either contain this value or be equal with it.

**contains**

an optional boolean. The default value is `true`. For the `true` value, the default value of the attribute must contain the `defaultAttributeValue` parameter. In case the value is `false`, the two values must be the same.

The `oxy:current-selected-element()` Function

This function returns the fully selected element. In case no element is selected, the function returns an empty sequence.

```xml
oxy:current-selected-element()[self::p]/b
```

This example returns the `b` elements that are children of the currently selected `p` element.

**Menu**

In the **Menu** sub-tab you configure what `framework` specific actions appear in the Oxygen XML Developer plugin menu. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Developer plugin menu.
plugin menu. To add an action in this section as a sibling of the currently selected action, use the Add as sibling button. To add an image in this section as a child of the currently selected action use the Add as child button.

The following actions are available in the Current actions section:

- **Edit**
  - Edits an item.
- **Remove**
  - Removes an item.
- **Move Up**
  - Moves an item up.
- **Move Down**
  - Moves an item down.

**Contextual menu**

In the Contextual menu sub-tab you configure what framework-specific action the Content Completion Assistant proposes. The sub-tab is divided in two sections: Available actions and Current actions.

The Available actions section presents a table that displays the actions defined in the Actions sub-tab, along with their icon, ID, and name. The Current actions section holds the actions that are displayed in the contextual menu of a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the Add as sibling button. To add an action in this section as a child of the currently selected action use the Add as child button.

The following actions are available in the Current actions section:

- **Edit**
  - Edits an item.
- **Remove**
  - Removes an item.
- **Move Up**
  - Moves an item up.
- **Move Down**
  - Moves an item down.

**Toolbar**

In the Toolbar sub-tab you configure what framework-specific action the Oxygen XML Developer plugin toolbar holds. The sub-tab is divided in two sections: Available actions and Current actions.

The Available actions section presents a table that displays the actions defined in the Actions sub-tab, along with their icon, ID, and name. The Current actions section holds the actions that are displayed in the Oxygen XML Developer plugin toolbar when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the Add as sibling button. To add an action in this section as a child of the currently selected action use the Add as child button.

The following actions are available in the Current actions section:

- **Edit**
  - Edits an item.
- **Remove**
  - Removes an item.
**Content Completion**

In the **Content Completion** sub-tab you configure what framework-specific the **Content Completion Assistant** proposes. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the Actions sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that the **Content Completion Assistant** proposes when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the **Add as sibling** button. To add an action in this section as a child of the currently selected action use the **Add as child** button.

The following actions are available in the **Current actions** section:

- **Edit**
  - Edits an item.

- **Remove**
  - Removes an item.

- **Move Up**
  - Moves an item up.

- **Move Down**
  - Moves an item down.

**The Templates Tab**

The **Templates** tab specifies a list of directories in which new file templates are located. These file templates are gathered from all the document types and presented in the **New** document dialog box.

**The Catalogs Tab**

The **Catalogs** tab specifies a list of XML catalogs which are added to all the catalogs that Oxygen XML Developer plugin uses to resolve resources.

**The Transformation Tab**

In the **Transformation** tab you configure the transformation scenarios associated with the framework you are editing. These are the transformation scenarios that are presented in the **Configure Transformation Scenarios** dialog box as associated with the type of the edited document.

You can set one or more of the scenarios from the **Transformation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog box and are also displayed on the tooltip of the **Apply transformation Scenario(s)**.

The **Transformation** tab offers the following options:

- **New**
  - Opens the **New scenario** dialog box allowing you to create a new transformation scenario.

- **Edit**
  - Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected transformation scenario.

- **Delete**
  - Deletes the currently selected transformation scenario.
The Validation Tab

In the Validation tab you configure the validation scenarios associated with the framework you are editing. These are the validation scenarios that are presented in the Configure Validation Scenarios dialog box as associated with the type of the edited document.

You can set one or more of the scenarios from the Validation tab as default. The scenarios set here as default are rendered bold in the Configure Transformation Scenarios dialog box and are also displayed on the tooltip of the Apply transformation Scenario(s) button.

The Validation tab offers the following options:

- **New**
  Opens the New scenario dialog box allowing you to create a new validation scenario.

- **Edit**
  Opens the Edit scenario dialog box allowing you to edit the properties of the currently selected validation scenario.

- **Delete**
  Deletes the currently selected validation scenario.

- **Import scenarios**
  Imports transformation scenarios.

- **Export selected scenarios**
  Export transformation scenarios.

- **Move Up**
  Moves the selection to the previous scenario.

- **Move Down**
  Moves the selection to the next scenario.

The Extensions Tab

The Extension tab specifies implementations of Java interfaces used to provide advanced functionality to the document type.


Editor Preferences

Oxygen XML Developer plugin lets you configure how the editor appears. To configure the appearance of the text editor, open the Preferences dialog box and go to Editor or right click in the editor window and choose Preferences.

The following options are available:

- **Editor background** - Sets the background color for both text editor and Diff Files editors.
Oxygen XML Developer plugin | Configuring Oxygen XML Developer plugin

- **Completion proposal background** - Sets the background color of the content completion window.
- **Completion proposal foreground** - Sets the foreground color of the content completion window.
- **Documentation window background** - Sets the background color of the documentation of elements suggested by the content completion assistant.
- **Documentation window foreground** - Sets the foreground color for the documentation of elements suggested by the content completion assistant.
- **Display quick-assist and quick-fix side hints** - Displays the Quick Assist and Quick Fix icon in the editor's left side line number stripe.
- **Line wrap** - Enables soft wrap of long lines, that is automatically wrap lines in edited documents. The document content is unaltered as the application does not use newline characters to break long lines.
  
  **Note**: When you enable the Line wrap option, Oxygen XML Developer plugin disables the Highlight current line option.

- **Highlight matching tag** - If you place the cursor on a start or end tag, Oxygen XML Developer plugin highlights the corresponding member of the pair. You can also customize the highlight color.

- **Beep on operation finished** - Oxygen XML Developer plugin emits a short beep when a validate, check well-formedness, or transform action has ended;
  
  **Note**: When the validation or the transformation process of a document is successful, the beep signal has a higher audio frequency, as opposed to when the validation fails, and the beep signal has a lower audio frequency. On the Windows platform, for other operations, the default system sound (Asterisk) is used. You can configure it by changing the sound theme.

- **Minimum fold range** - You can specify the minimum number of lines in a block for which the folding support becomes active. If you modify this value, the change takes effect next time you open / reopen the editor.

**Edit modes Preferences**

Oxygen XML Developer plugin lets you configure which edit mode a file is opened in the first time it is opened. This setting only affects the first time a file is opened. The current editing mode of each file is saved when the file is closed and restored the next time it is opened. To configure the Edit modes options, open the Preferences dialog box and go to Editor > Edit modes.

The initial edit mode can be one of the following:

- **Text**
- **Grid**
- Design (available only for the W3C XML Schema editor).

The Oxygen XML Developer plugin “Edit modes” Preferences Page
The Oxygen XML Developer plugin “Edit modes” Preferences Page

Text Diagram Preferences

For certain XML languages, Oxygen XML Developer plugin provides a diagram view as part of the text mode editor. To configure the Diagram preferences, open the Preferences dialog box and go to Editor > Edit modes > Text / Diagram.

The following options are available:

• Show Full Model XML Schema diagram - When this option is selected, the Text mode editor for XML Schemas is shown with a split screen view which shows a diagram of the schema structure. This may be useful for seeing the effect of schema changes you make in text view. For editing a the schema using diagram rather than text, use the schema Design view.

  Note: When handling very large schemas, displaying the schema diagram might affect the performance of your system. In such cases, disabling the schema diagram view improves the speed of navigation through the edited schema.

• Enable Relax NG diagram and related views - Enables the Relax NG schema diagram and synchronization with the related views (Attributes, Model, Elements, Outline).

• Show Relax NG diagram - Displays the Relax NG schema diagram in Full Model View and Logical Model View.

• Enable NVDL diagram and related views - Enables the NVDL schema diagram and synchronization with the related views (Attributes, Model, Elements, Outline).

• Show NVDL diagram - Displays the NVDL schema diagram in Full Model View and Logical Model View.

• Location relative to editor - Sets the location of the schema diagram panel relative to the diagram Text editor.

Grid Preferences

Oxygen XML Developer plugin provides a Grid view of an XML document. To configure the Grid options, open the Preferences dialog box and go to Editor > Edit modes > Grid.

The following options are available:

• Compact representation - If selected, the compact representation of the grid is used: a child element is displayed beside the parent element. In the non-compact representation, a child element is nested below the parent.

• Format and indent when passing from grid to text or on save - If selected, the content of the document is formatted and indented each time you switch from the Grid view to the Text view.

• Default column width (characters) - Sets the default width in characters of a table column of the grid. A column can hold:

  • element names
  • element text content
• attribute names
• attribute values

If the total width of the grid structure is too large you can resize any column by dragging the column margins with the mouse pointer, but the change is not persistent. To make it persistent, set the new column width with this option.

• **Active cell color** - Sets the background color for the active cell of the grid. There is only one active cell at a time. The keyboard input always goes to the active cell and the selection always contains it.
• **Selection color** - Background color for the selected cells of the grid except the active cell.
• **Border color** - The color used for the lines that separate the grid cells.
• **Background color** - The background color of grid cells that are not selected.
• **Foreground color** - The text color of the information displayed in the grid cells.
• **Row header colors - Background color** - The background color of row headers that are not selected.
• **Row header colors - Active cell color** - The background color of the row header cell that is currently active.
• **Row header colors - Selection color** - The background color of the header cells corresponding to the currently selected rows.
• **Column header colors - Background color** - The background color of column headers that are not selected.
• **Column header colors - Active cell color** - The background color of the column header cell that is currently active.
• **Column header colors - Selection color** - The background color of the header cells corresponding to the currently selected columns.

The column headers are painted with two color gradients, one for the upper 1/3 part of the header and the other for the lower 2/3 part. The start and end colors of the first gradient are set with the first two color buttons. The start and end colors of the second gradient are set with the last two color buttons.

**Schema Design Preferences**

Oxygen XML Developer plugin provides a graphical schema design editor to make editing XML schemas easier. To configure the **Schema Design** options, open the Preferences dialog box and go to **Editor > Edit modes > Schema Design**.

The following options are available in the **Schema Design** preferences page:

• **Show annotation in the diagram** - When selected, Oxygen XML Developer plugin displays the content of xs:documentation elements in the XML Schema Design view.
• **When trying to edit components from another schema** - The schema diagram editor will combine schemas imported by the current schema file into a single schema diagram. You can choose what happens if you try to edit a component from an imported schema. The options are:
  • **Always go to its definition** - Oxygen XML Developer plugin opens the imported schema file so that you can edit it.
  • **Never go to its definition** - The imported schema file is not opened. The definition cannot be edited in place.
  • **Always ask** - Oxygen XML Developer plugin asks if you want to open the imported schema file.

**Properties**

Oxygen XML Developer plugin lets you control which properties to display for XML Schema components in the XML Schema Design view. To configure the schema design properties displayed, open the Preferences dialog box and go to **Editor > Edit modes > Schema Design > Properties**.

The available options are:

• **Show additional properties in the diagram** - If selected, the properties selected in the property table are shown in the XML Schema Diagram view. This option is selected by default.
• The properties that can be selected. In the table, select those properties you want to be displayed. You can also select if you want the property to be displayed only when it is actually defined in the schema.
**Text Diagram Preferences**

For certain XML languages, Oxygen XML Developer plugin provides a diagram view as part of the text mode editor. To configure the Diagram preferences, open the Preferences dialog box and go to Editor > Edit modes > Text / Diagram.

The following options are available:

- **Show Full Model XML Schema diagram** - When this option is selected, the Text mode editor for XML Schemas is shown with a split screen view which shows a diagram of the schema structure. This may be useful for seeing the effect of schema changes you make in text view. For editing a the schema using diagram rather than text, use the schema Design view.

  **Note:** When handling very large schemas, displaying the schema diagram might affect the performance of your system. In such cases, disabling the schema diagram view improves the speed of navigation through the edited schema.

- **Enable Relax NG diagram and related views** - Enables the Relax NG schema diagram and synchronization with the related views (Attributes, Model, Elements, Outline).

- **Show Relax NG diagram** - Displays the Relax NG schema diagram in Full Model View and Logical Model View.

- **Enable NVDL diagram and related views** - Enables the NVDL schema diagram and synchronization with the related views (Attributes, Model, Elements, Outline).

- **Show NVDL diagram** - Displays the NVDL schema diagram in Full Model View and Logical Model View.

- **Location relative to editor** - Sets the location of the schema diagram panel relative to the diagram Text editor.

**Format Preferences**

This preferences page contains various formatting options which influence editing and formatting in the Text mode.

**Note:** These settings apply to the formatting of source documents. The formatting of output documents is determined by the transformation scenarios that create them.

The following options are available:

- **Detect indent on open** - Oxygen XML Developer plugin detects how a document is indented when it is opened. Oxygen XML Developer plugin uses a heuristic method of detection by computing a weighted average indent value from the initial document content. You can disable this setting if the detected value does not work for your particular case and you want to use a fixed-size indent for all the edited documents.

  **Tip:** If you want to minimize the formatting differences created by the Format and Indent operation in a document edited in the Text edited mode, make sure that both the Detect indent on open and Detect line width on open options are enabled.

- **Use zero-indent, if detected** - by default, if no indent was detected in the document, the fixed-size indent is used. Enable this option if all your document have no indentation and you want to keep them that way.

- **Indent with tabs** - If selected, indents are created using tab characters. If unchecked, lines are indented using space characters.

- **Indent size** - A fixed number of spaces used for indenting a line.

- **Hard line wrap (Limit to "Line width - Format andIndent")** - If selected, when typing content in the Text editing mode when the maximum line width is reached, a line break is automatically inserted.

- **Indent on enter** - If disabled, when you press the Enter key to insert a line break in the Text editing mode, no indentation will be added to the new line.

- **Enable smart enter** - If selected, when you press the Enter key between a start and an end XML tag in the Text editing mode, the cursor is placed in an indented position on the empty line formed between the start and end tag.

- **Detect line width on open** - When selected, Oxygen XML Developer plugin detects the line width automatically when the document is opened.

- **Format and indent the document on open** - When selected, an XML document is formatted and indented before opening it in Oxygen XML Developer plugin.
• **Line width - Format and Indent** - Defines the number of characters after which the **Format and Indent** (pretty-print) action performs hard line wrapping. For example, if set to 100, after a **Format and Indent** action, the longest line will have at most 100 characters. This setting is also used when saving the XML content edited in the **Author** editing mode.

• **Clear undo buffer before Format and Indent** - The **Format and Indent** operation can be undone, but if used intensively, a considerable amount of the memory allocated for Oxygen XML Developer plugin will be used for storing the undo states. If this option is selected, Oxygen XML Developer plugin empties the undo buffer before doing a **Format and Indent** operation. This means you will not be able to undo any changes you made before the format and indent operation. Select this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

The indent size and line width limit settings are used in various places in the application:

• When the **Format and Indent** action is used in the **Text** editing mode.
• When you press **ENTER** in the **Text** editing mode to break a line.
• When editing in the **Text** mode with **Hard line wrap** enabled.

To watch our video demonstration about the formatting options offered by Oxygen XML Developer plugin, go to [http://oxygenxml.com/demo/Autodetect_Formating.html](http://oxygenxml.com/demo/Autodetect_Formating.html).

**XML Formatting Preferences**

To configure the XML Formatting options, **open the Preferences dialog box** and go to **Editor > Format > XML**.

The following options are available:

• **Preserve empty lines** - The **Format and Indent** operation preserves all empty lines found in the document.
• **Preserve text as it is** - The **Format and Indent** operation preserves text content as it is, without removing or adding any white space.
• **Preserve line breaks in attributes** - Line breaks found in attribute values are preserved.
  
  Note: When this option is enabled, the **Break long attributes** option is automatically disabled.

• **Break long attributes** - The **Format and Indent** operation breaks long attribute values.
• **Indent inline elements** - The **inline elements** are indented on separate lines if they are preceded by white spaces and they follow another element start or end tag. Example:

  **Original XML**:

  ```xml
  <root>
  text <parent> <child/></child> </parent>
  </root>
  ```

  **Indent inline elements enabled**:

  ```xml
  <root> text <parent>
  <child/>
  </parent>
  </root>
  ```

  **Indent inline elements disabled**:

  ```xml
  <root> text <parent> <child/> </parent> </root>
  ```

• **Expand empty elements** - The **Format and Indent** operation outputs empty elements with a separate closing tag (for example, `<a atr1="v1">`/`</a>`). When not enabled, the same operation represents an empty element in a more compact form (`<a atr1="v1"/>`).
• **Sort attributes** - The **Format and Indent** operation sorts the attributes of an element alphabetically.
• **Add space before slash in empty elements** - Inserts a space character before the trailing `/` and `>` of empty elements.
• **Break line before attribute's name** - The **Format and Indent** operation breaks the line before the attribute name.
• **Element spacing** - Controls how the application handles whitespaces found in XML content:
• **Preserve space** - List of elements for which the **Format and Indent** operation preserves the whitespaces (such as blanks, tabs, and newlines). The elements can be specified by name or by XPath expressions:
  - `elementName`
  - `//elementName`
  - `/elementName1/elementName2/elementName3`
  - `//xs:localName`

  The namespace prefixes (such as `xs`) are treated as part of the element name without taking its binding to a namespace into account.

• **Default space** - The list contains the names of the elements for which the content is normalized (multiple contiguous whitespaces are replaced by a single space), before applying the **Format and Indent** operation.

• **Mixed content** - The elements from this list are treated as mixed content when applying the **Format and Indent** operation. The lines are split only when whitespaces are encountered.

• **Schema aware format and indent** - The **Format and Indent** operation takes into account the schema information regarding the `space preserve`, `mixed`, or `element only` properties of an element.

• **Indent (when typing) in preserve space elements** - Normally, the `Preserve space` elements (identified by the `xml:space` attribute set to `preserve` or by their presence in the `Preserve space` elements list) are ignored by the **Format and Indent** operation. When this option is enabled and you edit one of these elements, its content is formatted.

• **Indent on paste** - **sections with number of lines less than 300** - When you paste a chunk of text that has less than 300 lines, the inserted content is indented. To keep the original indent style of the document you copy content from, disable this option.

**Whitespace Preferences**

Oxygen XML Developer plugin lets you configure which Unicode space characters are treated as space characters when normalizing whitespace in XML documents. To configure the **Whitespace** preferences, open the Preferences dialog box and go to **Editor > Format > XML > Whitespaces**.

This table lists the Unicode whitespace characters. Check any that you want to have treated as whitespace when formatting and indenting an XML document.

The whitespaces are normalized when the **Format and Indent** action is applied on an XML document.

The characters with the codes 9 (TAB), 10 (LF), 13 (CR) and 32 (SPACE) are always considered to be whitespace characters and cannot be deselected.

**XQuery Formatting Preferences**

To configure the **XQuery** Formatting options, open the Preferences dialog box and go to **Editor > Format > XQuery**.

The following options are available:

• **Preserve line breaks** - All initial line breaks are preserved.

• **Break line before attribute’s name** - Each attribute of an XML element is written on a new line and properly indented.

**XPath Formatting Preferences**

To configure the **XPath** Formatting options, open the Preferences dialog box and go to **Editor > Format > XPath**.

The following option is available:

• **Format XPath code embedded in XSLT, XSD and Schematron files** - If enabled, the **Format and Indent** action applied on a XSD, XSLT, or Schematron document will perform an XPath-specific formatting on the values of the attributes that accept XPath expressions.

  **Note:** For XSLT documents, the formatting is not applied to `attribute value templates`.

---

Oxygen XML Developer plugin  | Configuring Oxygen XML Developer plugin | 508
CSS Properties Formatting Preferences

Oxygen XML Developer plugin can format and indent your CSS files. To configure the CSS Format options, go to Editor > Format > CSS.

The following options control how your CSS files are formatted and indented:

- **Indent class content** - The class content is indented. Enabled by default.
- **Class body on new line** - The class body (including the curly brackets) is placed on a new line.
- **Add new line between classes** - An empty line is added between two classes.
- **Preserve empty lines** - The empty lines from the CSS content are preserved.
- **Allow formatting embedded CSS** - The CSS content embedded in XML is formatted when the XML content is formatted.

JavaScript Properties Formatting Preferences

To configure the JavaScript format options, open the Preferences dialog box and go to Editor > Format > JavaScript.

The following options control the behavior of the Format and Indent action:

- **Start curly brace on new line** - Opening curly braces start on a new line.
- **Preserve empty lines** - Empty lines in the JavaScript code are preserved. This option is enabled by default.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows Oxygen XML Developer plugin to format embedded JavaScript code, taking precedence over the Schema aware format and indent option. This option is enabled by default.

Content Completion Preferences

Oxygen XML Developer plugin provides a Content Completion Assistant that list available options at any point in a document and can auto-complete structures, elements, and attributes. These options control how the Content Completion Assistant works.

To configure the Content Completion preferences, open the Preferences dialog box and go to Editor > Content Completion.

The following options are available:

- **Auto close the last opened tag** - Oxygen XML Developer plugin closes the last open tag when you type </.
- **Automatically rename/delete/comment matching tag** - If you rename, delete, or comment out a start tag, Oxygen XML Developer plugin automatically renames, deletes, or comments out the matching end tag.

  **Note:** If you select Toggle comment for multiple starting tags and the matching end tags are on the same line as other start tags, the end tags are not commented.

- **Use content completion** - Turns content completion on or off.
- **Close the inserted element** - When you choose an entry from the Content Completion Assistant list of proposals, Oxygen XML Developer plugin inserts both start and end tags.

  - **If it has no matching tag** - The end tag of the inserted element is automatically added only if it is not already present in the document.
  - **Add element content** - Oxygen XML Developer plugin inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is associated with the edited XML document.

  - **Add optional content** - Oxygen XML Developer plugin inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
  - **Add first Choice particle** - Oxygen XML Developer plugin inserts the first choice particle specified in the DTD, XML Schema, or RELAX NG schema.

- **Case sensitive search** - When enabled, the search in the content completion assistant window when you type a character is case-sensitive (’a’ and ’A’ are different characters).
Note: This option is ignored when the current language itself is not case sensitive. For example, the case is ignored in the CSS language.

- **Cursor position between tags** - When selected, Oxygen XML Developer plugin automatically moves the cursor between start and end tag after inserting the element. This only applies to:
  - Elements with only optional attributes or no attributes at all.
  - Elements with required attributes, but only when the **Insert the required attributes** option is disabled.

- **Show all entities** - Oxygen XML Developer plugin displays a list with all the internal and external entities declared in the current document when the user types the start character of an entity reference (i.e. `&`).

- **Insert the required attributes** - Oxygen XML Developer plugin inserts automatically the required attributes taken from the DTD or XML Schema.

- **Insert the fixed attributes** - Oxygen XML Developer plugin automatically inserts any `FIXED` attributes from the DTD or XML Schema for an element inserted with the help of the **Content Completion Assistant**.

- **Show recently used items** - When checked, Oxygen XML Developer plugin remembers the last inserted items from the **Content Completion Assistant** window. The number of items to be remembered is limited by the **Maximum number of recent items shown** list box. These most frequently used items are displayed on the top of the content completion window their icon is decorated with a small red square.

- **Maximum number of recent items shown** - limits the number of recently used items presented at the top of the **Content Completion Assistant** window.

- **Learn attributes values** - Oxygen XML Developer plugin learns the attribute values used in a document.

- **Learn on open document** - Oxygen XML Developer plugin automatically learns the document structure when the document is opened.

- **Learn words** (Dynamic Abbreviations, available on **Ctrl Space (Command Space on OS X)**) - When selected, Oxygen XML Developer plugin learns the typed words and makes them available in a content completion fashion by pressing **Ctrl Space (Command Space on OS X)** on your keyboard.

  Note: In order to be learned, the words need to be separated by space characters.

### Annotations Preferences

Different types of schemas (XML Schema, DTDs, Relax NG) can include annotations that document the various elements and attributes that they define. Oxygen XML Developer plugin can display these annotations when offering content completion suggestions. To configure the **Annotations** preferences, open the **Preferences dialog box** and go to **Editor > Content Completion > Annotations**.

The following options are available:

- **Show annotations in Content Completion Assistant** - Oxygen XML Developer plugin displays the schema annotations of an element, attribute, or attribute value currently selected in the **Content Completion Assistant** proposals list.

- **Show annotations in tooltip** - Oxygen XML Developer plugin displays the annotation of elements and attributes as a tooltip when you hover over them with the cursor in the editing area or in the **Elements** view.

- **Show annotation in HTML format, if possible** - This option allows you to view the annotations associated with an element or attribute in HTML format. It is available when editing XML documents that have associated an XML Schema or Relax NG schema. When this option is disabled the annotations are converted and displayed as plain text.

- **Prefer DTD comments that start with "doc:" as annotations** - To address the lack of dedicated annotation support in DTD documents, Oxygen XML Developer plugin recommends prefixing with the `doc:` particle all comments intended to be shown to the developer who writes an XML validated against a DTD schema.

When this option is enabled, Oxygen XML Developer plugin uses the following mechanism to collect annotations:

- if at least one `doc:` comment is found in the entire DTD, only comments of this type are displayed as annotations
- if no `doc:` comment is found in the entire DTD, all comments are considered annotations and displayed as such

When the option is disabled, all comments, regardless of their type, are considered annotations and displayed as such.
• **Use all Relax NG annotations as documentation** - When this option is selected, any element outside the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0`, is considered annotation and is displayed in the annotation window next to the **Content Completion Assistant** window and in the **Model** view. When this option is not selected, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

**XSL Preferences**

XSL stylesheets are often used to create output in XHTML or XSL-FO. In addition to suggesting content completion options for XSLT stylesheet elements, Oxygen XML Developer plugin can suggest elements from these vocabularies. To configure the XSL content completion options, *open the Preferences dialog box* and go to **Editor > Content Completion > XSL**.

The following options are available:

• **Automatically detect XHTML transitional or Formatting objects** - Detects if the output being generated is XHTML or FO and provides content completion for those vocabularies. Oxygen XML Developer plugin analyzes the namespaces declared in the root element to find an appropriate schema.

  If the detection fails, Oxygen XML Developer plugin uses one of the following options:

  • **None** - The **Content Completion Assistant** suggests only XSLT elements.
  • **XHTML transitional** - The **Content Completion Assistant** includes XHTML Transitional elements as substitutes for `xsl:element`.
  • **Formatting objects** - The **Content Completion Assistant** includes Formatting Objects (XSL-FO) elements as substitutes for `xsl:element`.
  • **Custom schema** - If you want content completion hints for a different output vocabulary, enter the path to the schema for that vocabulary here. Supported schema types are DTD, XML Schema, RNG schema, or NVDL schema for inserting elements from the target language of the stylesheet.

You can choose an additional schema that will be used for documenting XSL stylesheets. Either select the built-in schema or choose a custom one. Supported schema types are XSD, RNG, RNC, DTD, and NDVL.

**XPath Preferences**

Oxygen XML Developer plugin provides content-completion support for XPath expressions. To configure the options for the content completion in XPath expressions, *open the Preferences dialog box* and go to **Editor > Content Completion > XPath**.

The following options are available:

• **Enable content completion for XPath expressions** - Enables the **Content Completion Assistant** in XPath expressions that you enter in the `match`, `select`, and `test` XSL attributes and also in the XPath toolbar.

  • **Include XPath functions** - When this option is selected, XPath functions are included in the content completion suggestions.
  • **Include XSLT functions** - When this option is selected, XSLT functions are included in the content completion suggestions.
  • **Include axes** - When this option is selected, XSLT axes are included in the content completion suggestions.
  • **Show signatures of XSLT / XPath functions** - Makes the editor indicate the signature of the XPath function located at the caret position in a tooltip. See the **XPath Tooltip Helper** section for more information.

**XSD Preferences**

Oxygen XML Developer plugin provides content completion assistance when you are writing an XML Schema (XSD). To configure XSD preferences, *open the Preferences dialog box* and go to **Editor > Content Completion > XSD**. The options in these preferences page define what elements are suggested by the **Content Completion Assistant**, in addition to the ones from the XML Schema (defined by the `xs:annotation/xs:appinfo` elements).

The following options are available:

• **None** - The **Content Completion Assistant** offers only the XML Schema schema information.
• **ISO Schematron** - The **Content Completion Assistant** includes ISO Schematron elements in `xs:appinfo`.
• **Schematron 1.5** - The **Content Completion Assistant** includes Schematron 1.5 elements in `xs:appinfo`.
• **Other** - The **Content Completion Assistant** includes in `xs:appinfo` elements from an XML Schema identified by an URL.

**Syntax Highlight Preferences**

Oxygen XML Developer plugin supports syntax highlighting of XML in the *Text mode* editor, DTD, Relax NG (XML and Compact Syntax), Java, JavaScript / JSON, PHP, CSS, XQuery, C++, C, Perl, Properties, SQL, Shell and Batch documents.

To configure syntax highlighting, *open the Preferences dialog box* and go to **Editor > Syntax Highlight**.

To set syntax colors for a language, expand the listing for that language in the top panel to show the list of syntax items for that language. Use the color and style selectors to change how each syntax item is displayed. The results of your changes are shown in the preview panel. If you do not know the name of the syntax token that you want to configure, click on that token in the **Preview** area to select it.

**Note:** All default color sets come with a high-contrast variant, which is automatically used when you switch to a black-background or white-background high-contrast theme in your Windows operating system settings. The high-contrast theme will not overwrite any default color you set in **Syntax Highlight** preferences page.

The settings for XML documents are used also in XSD, XSL, RNG documents. The **Preview** area has separate tabs for XML, XSD, XSL, RNG.

The **Enable nested syntax highlight** option controls if different content types mixed in the same file (like PHP, JS and CSS scripts inside an HTML file) are highlighted according with the color schemes defined for each content type.

**Elements / Attributes by Prefix Preferences**

Oxygen XML Developer plugin lets you specify different colors for XML elements and attributes with specific namespace prefixes. To configure the **Elements / Attributes by Prefix** preferences, *open the Preferences dialog box* and go to **Editor > Syntax Highlight > Elements / Attributes by Prefix**.

To change the syntax coloring for a specific namespace prefix, choose the prefix from the list, or add a new one using the **New** button, and use the color and style selectors to set the syntax highlighting style for that namespace prefix.

**Note:** Syntax highlighting is based on the literal namespace prefix, not the namespace that the prefix is bound to in the document.

If you want only the prefix, and not the whole element or attribute name, to be styled differently, select **Draw only the prefix with a separate color**.

**Open / Save Preferences**

Oxygen XML Developer plugin lets you control how files are opened and saved. To configure the **Open / Save** options, *open the Preferences dialog box* and go to **Editor > Open / Save**.

**Open**

The following options apply to opening files:

• **Format document when longest line exceeds** - Oxygen XML Developer plugin will create line breaks if the characters in a line exceed the specified value. You can choose one of the following:
  • Always format
  • Never format
  • Always ask

**Save**

The following options apply to saving files:
• **Check errors on save** - If enabled, Oxygen XML Developer plugin runs a validation that checks your document for errors before saving it.

• **Save all files before transformation or validation** - Saves all open files before validating or transforming an XML document. This ensures that any dependencies are resolved when modifying the XML document and its XML Schema.

**Performance**

The following options cover performance issues when dealing with large files:

• **Clear undo buffer on save** - If selected, Oxygen XML Developer plugin clears its undo buffer when you save a document. Therefore, modifications made prior to saving the document cannot be undone. Select this option if you frequently encounter *out of memory* errors when editing large documents.

**Save Hooks Preferences**

Oxygen XML Developer plugin includes an option for automatically compiling LESS stylesheets. To set this option, *open the Preferences dialog box* and go to **Editor > Open / Save > Save hooks**.

The following option can be enabled or disabled:

• **Automatically compile LESS to CSS when saving** - If enabled, when you save a LESS file it will automatically be compiled to CSS (disabled by default).

  ! Important: If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

**Templates Preferences**

This page groups the preferences for code templates and document templates:

• **Code Templates**

• **Document Templates**

**Code Templates Preferences**

*Code templates* are code fragments that can be inserted at the current editing position. Oxygen XML Developer plugin comes with a set of built-in templates for CSS, LESS, Schematron, XSL, XQuery, and XML Schema document types. You can also define your own code templates and share them with your colleagues using the template export and import functions.

To configure **Code Templates**, *open the Preferences dialog box* and go to **Editor > Templates > Code Templates**.

This preferences page contains a list of all the available code templates (both built-in and custom created ones) and a code preview area. You can disable any code template by deselecting it.

The following actions are available:

**New**

Opens the *Code template* dialog that allows you to define a new code template. You can define the following fields:

• **Name** - The name of the code template.

• **Description** - The description of the code template. that will appear in the **Code Templates** preferences page and in the tooltip message when selecting it from the **Content Completion Assistant**.

• **Associate with** - You can choose to set the code template to be associated with a specific type of editor or for all editor types.

• **Shortcut key** - Allows you to configure a shortcut key that can be used to insert the code template. The + character separates keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut is platform-independent and the following modifiers are used:

  • M1 represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
• M2 represents the **Shift** key.
• M3 represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
• M4 represents the **Ctrl** key on MacOS X, and is undefined on other platforms.

• **Content** - Text box where you define the content that is used when the code template is inserted.

**Edit**

Opens the **Code template** dialog and allows you to edit an existing code template. You can edit the following fields:

• **Description** - The description of the code template, that will appear in the **Code Templates** preferences page and in the tooltip message when selecting it from the **Content Completion Assistant**.

• **Shortcut key** - Allows you to configure a shortcut key that can be used to insert the code template. The + character separates keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut is platform-independent and the following modifiers are used:
  • M1 represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
  • M2 represents the **Shift** key.
  • M3 represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
  • M4 represents the **Ctrl** key on MacOS X, and is undefined on other platforms.

• **Content** - Text box where you define the content that is used when the code template is inserted.

**Duplicate**

Creates a duplicate of the currently selected code template.

**Delete**

Deletes the currently selected code template. This action is disabled for the built-in code templates.

**Export**

Exports a file with code templates.

**Import**

Imports a file with code templates that was created by the **Export** action.

You can use the following **editor variables** when you define a code template in the **Content** text box:

• ${caret} - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.

• ${selection} - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.

• ${ask('message', type, ('real_value1': 'rendered_value1'; 'real_value2': 'rendered_value2'; ...), 'default_value')} - To prompt for values at runtime, use the `ask('message', type, ('real_value1': 'rendered_value1'; 'real_value2': 'rendered_value2'; ...), 'default_value')` editor variable. You can set the following parameters:
  • 'message' - The displayed message. Note the quotes that enclose the message.
  • type - Optional parameter, with one of the following values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Format: ${ask('message', url, 'default_value')}</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Description: Input is considered a URL. Oxygen XML Developer plugin checks that the provided URL is valid.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Input URL', url)} - The displayed dialog box has the name Input URL. The expected input type is URL.</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Input URL', url, '<a href="http://www.example.com')%7D">http://www.example.com')}</a> - The displayed dialog box has the name Input URL. The expected input type is URL. The input field displays the default value <a href="http://www.example.com">http://www.example.com</a>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Format: ${ask('message', value, 'default')}</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>password</td>
<td>Description: The input is hidden with bullet characters.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Input password', password)} - The displayed dialog box has the name 'Input password' and the input is hidden with bullet symbols.</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Input password', password, 'abcd')} - The displayed dialog box has the name 'Input password' and the input hidden with bullet symbols. The input field already contains the default abc value.</td>
</tr>
<tr>
<td>generic</td>
<td>Description: The input is considered to be generic text that requires no special handling.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Hello world!')} - The dialog box has a Hello world! message displayed.</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Hello world!', generic, 'Hello again!')} - The dialog box has a Hello world! message displayed and the value displayed in the input box is 'Hello again!'.</td>
</tr>
<tr>
<td>relative_url</td>
<td>Description: Input is considered a URL. Oxygen XML Developer plugin tries to make the URL relative to that of the document you are editing.</td>
</tr>
<tr>
<td></td>
<td>Note: If the $ask editor variable is expanded in content that is not yet saved (such as an untitled file, whose path cannot be determined), then Oxygen XML Developer plugin will transform it into an absolute URL.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>• ${ask('File location', relative_url, 'C:/example.txt')} - The dialog box has the name 'File location'. The URL inserted in the input box is made relative to the current edited document location.</td>
</tr>
<tr>
<td>combobox</td>
<td>Description: Displays a dialog box that offers a drop-down list. The drop-down list is populated with the given rendered_value values. Choosing such a value will return its associated value (real_value).</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>• ${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX', 'osx'))} - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems. The associated value will be returned based upon your selection.</td>
</tr>
<tr>
<td></td>
<td>Note: In this example Mac OS X is the default selected value and if selected it would return osx for the output.</td>
</tr>
</tbody>
</table>
### editable_combobox

**Format:**

```latex
${\text{ask('message', editable_combobox, 
\{\text{'real_value1':'rendered_value1';...;\text{'real_valueN':'rendered_valueN'}\}, \text{'default'}\})}
```

**Description:**

Displays a dialog box that offers a drop-down list with editable elements. The drop-down list is populated with the given rendered_value values. Choosing such a value will return its associated real value (real_value) or the value inserted when you edit a list entry.

**Example:**

- `\${\text{ask('Operating System', editable_combobox, 
\{\text{'win':'Microsoft Windows';\text{'osx':'Mac OS X';\text{'lnx':'Linux/UNIX'}\}, \text{'osx'}\})}`
  - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.

### radio

**Format:**

```latex
${\text{ask('message', radio, 
\{\text{'real_value1':'rendered_value1';...;\text{'real_valueN':'rendered_valueN'}\}, \text{'default'}\})}
```

**Description:**

Displays a dialog box that offers a series of radio buttons. Each radio button displays a rendered_value and will return an associated real_value.

**Example:**

- `\${\text{ask('Operating System', radio, 
\{\text{'win':'Microsoft Windows';\text{'osx':'Mac OS X';\text{'lnx':'Linux/UNIX'}\}, \text{'osx'}\})}`
  - The dialog box has the name 'Operating System'. The radio button group allows you to choose between the three operating systems.
  - **Note:** In this example Mac OS X is the default selected value and if selected it would return osx for the output.

- `\text{'default-value'} - optional parameter. Provides a default value.
- `\text{'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.
- `\text{'uuid'} - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java UUID class.
- `\text{'id'} - Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- `\text{'cfn'} - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- `\text{'cfne'} - Current file name with extension. The current file is the one currently opened and selected.
- `\text{'cf'} - Current file as file path, that is the absolute file path of the current edited document.
- `\text{'cfd'} - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- `\text{'frameworksDir'} - The path (as file path) of the [OXYGEN_DIR]/frameworks directory.
- `\text{'pd'} - Current project folder as file path. Usually the current folder selected in the Project View.
- `\text{'oxygenInstallDir'} - Oxygen XML Developer plugin installation folder as file path.
- `\text{'homeDir'} - The path (as file path) of the user home folder.
- `\text{'pn'} - Current project name.
• \${env(VAR_NAME)} - Value of the VAR_NAME environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the \${system(var.name)} editor variable.

• \${system(var.name)} - Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as -Dvar.name=var.value. If you are looking for operating system environment variables, use the \${env(VAR_NAME)} editor variable instead.

• \${date(pattern)} - Current date. The allowed patterns are equivalent to the ones in the Java SimpleDateFormat class. Example: yyyy-MM-dd;

Note: This editor variable supports both the xs:date and xs:dateTime parameters. For details about xs:date, go to http://www.w3.org/TR/xmlschema-2/#date. For details about xs:dateTime, go to http://www.w3.org/TR/xmlschema-2/#dateTime.

Document Templates Preferences

Oxygen XML Developer plugin provides a selection of document templates that make it easier to create new documents in a variety of formats. The list of available templates is presented when you create a new document. You can add your own templates to this list by creating template files in a directory and adding that directory to the list of template directories that Oxygen XML Developer plugin uses. To add a template directory, open the Preferences dialog box and go to Editor > Templates > Document Templates.

You can add new document template location folders and manage existing ones. You can also alter the order in which Oxygen XML Developer plugin looks into these directories by using the Up and Down buttons.

Spell Check Preferences

Oxygen XML Developer plugin provides spell check support in the text edit mode. To configure the Spell Check options, open the Preferences dialog box and go to Editor > Spell Check.

The following options are available:

• Spell checking engine - Oxygen XML Developer plugin ships with two spell check engines, Hunspell and Java spell checker. The two engines come with different dictionaries. When you select an engine here, the list of languages in the Default language option changes based on the available dictionaries for the engine you have chosen.

• Automatic Spell Check - When selected, Oxygen XML Developer plugin checks spelling as you type and highlights misspelled words in the document.

• Select editors - You can select which editors (and therefore which file types) will be automatically spelled checked. File types for which automatic spell check is generally not useful, like CSS and DTD, are excluded by default.

Language Options Section

• Default language - The default language list allows you to choose the language used by the spell check engine when the language is not specified in the source file. You can add additional dictionaries to the spell check engines.

• Use "lang" and "xml:lang" attributes - When this option is selected, the contents of an element with one of the lang or xml:lang attributes is checked in that language. When this option is enabled, choose between the following two options for instances when these attributes are missing:

  • Use the default language - If the lang and xml:lang attributes are missing, the selection in the Default language list is used.

  • Do not check - If the lang and xml:lang attributes are missing, the element is not checked.

XML Spell Checking in Section

You can choose to check the spelling inside the following XML items:

• Comments

• Attribute values

• Text
Options Section

- **Check capitalization** - When selected, the spell checker reports capitalization errors, for example a word that starts with lowercase after *etc.* or *i.e.*.
- **Check punctuation** - When selected, the spell checker checks punctuation. Misplaced white space and unusual sequences, like a period following a comma, are highlighted as errors.
- **Ignore mixed case words** - When selected, the spell checker does not check words containing mixed case characters (for example, *SpellChecker*).
- **Ignore acronyms** - When selected, acronyms are not reported as errors.
- **Ignore words with digits** - When selected, the spell checker does not check words containing digits (for example, *b2b*).
- **Ignore duplicates** - When selected, the spell checker does not signal two successive identical words as an error.
- **Ignore URL** - When selected, the spell checker ignores words looking like URLs or file names (for example, *www.oxygenxml.com* or *c:\boot.ini)*.
- **Allow compounds words** - When selected, all words formed by concatenating two legal words with a hyphen (hyphenated compounds) are accepted. If recognized by the language, two words concatenated without hyphen (closed compounds) are also accepted.
- **Allow general prefixes** - When selected, a word formed by concatenating a recognized prefix and a legal word is accepted. For example if *mini-* is a registered prefix, the spell check engine accepts the word *mini-computer*.
- **Allow file extensions** - When selected, the spell checker accepts any word ending with recognized file extensions (for example, *myfile.txt* or *index.html*).

Ignore Elements Section

You can use the **Add** and **Remove** buttons to configure a list of element names or XPath expressions to be ignored by the spell checker. The following restricted set of XPath expressions are supported:

- `/` and `//` separators
- `*` wildcard

An example of an allowed XPath expression is: `/a/*/b`.

To change the color used by the spell check engine to highlight spelling errors, go to **Window (Eclipse on Mac OSX)** and choose **Preferences**. Then go to **General > Editors > Text Editors > Annotations > Spell Check Annotation**.

Spell Check Dictionaries Preferences

To set the Dictionaries preferences, **open the Preferences dialog box** and go to **Editor > Spell Check > Dictionaries**. This page allows you to configure the dictionaries and term lists (.tdi files) that Oxygen XML Developer plugin uses and to choose where to save new learned words.

The following options are valid when Oxygen XML Developer plugin uses the Hunspell spell checking engine:

- **Dictionaries and term lists default folder** - Displays the default location where the dictionaries and term lists that Oxygen XML Developer plugin uses are stored.
- **Include dictionaries and term list from** - Specifies a location where you can store dictionaries and term lists that are different from the default one.

**Note:** The spell checker takes into account dictionaries and term lists collected both from the default and custom locations and multiple dictionaries and term lists from the same language are merged into generic ones (for example, *en_UK.dic* from the default location is merged with *en_US.dic* from a custom location, and the result is that a third file is created for a generic dictionary called *en.dic*). However, if there is already a generic dictionary (for example, *en.dic*) saved in either the default or custom location, the other specific dictionaries (for example, *en_UK.dic* and *en_US.dic*) will not be merged and the existing generic dictionary will simply be used. Also, if the additional location contains a file with the same name as one from the default location, the file in the additional location takes precedence over the file from the default location.
• **Save learned words in the following location** - Specifies the target where the newly learned words are saved. By default, the target is the application preferences folder, but you can also choose a custom location.

• **Delete learned words** - Opens the list of learned words, allowing you to select the items you want to remove, without deleting the dictionaries and term lists.

**Document Checking Preferences**

To configure the **Document Checking** options, open the *Preferences* dialog box and go to Editor > Document Checking. This preferences page contains preferences for configuring how a document is checked for both well-formedness errors and validation errors.

The following options are available:

• **Maximum number of validation highlights** - If validation generates more errors than the number from this option only the first errors up to this number are highlighted in editor panel and on stripe that is displayed at right side of editor panel. This option is applied both for automatic validation and manual validation.

• **Clear validation markers on close** - If this option is selected all the error markers added in the Problems view for that document are removed when a document edited with the Oxygen XML Developer plugin plugin is closed.

• **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen XML Developer plugin.

• **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

**Mark Occurrences Preferences**

To configure the **Mark Occurrences** options, open the *Preferences* dialog box and go to Editor > Mark Occurrences:

The following preferences are available in this preferences page:

• **XML files** - Activates the Highlight IDs Occurrences feature in XML files.

• **XSLT files** - Activates the Highlight Component Occurrences feature in XSLT files.

• **XML Schema files and WSDL files** - Activates the Highlight Component Occurrences feature in XSD and WSDL files.

• **RNG files** - Activates the highlight component occurrences feature in RNG files.

• **Schematron files** - Activates the Highlight Component Occurrences feature in Schematron files.

• **Declaration highlight color** - Color used to highlight the component declaration.

• **Reference highlight color** - Color used to highlight component references.

**Custom Validation Engines Preferences**

To configure the options for Custom Validation Engines, open the *Preferences* dialog box and go to Editor > Custom Validations.

If you want to add a new custom validation tool or edit the properties of an exiting one you can use the Custom Validator dialog displayed by pressing the New button or the Edit button.
The configurable parameters of a custom validator are as follows:

- **Name** - Name of the custom validation engine that will be displayed in the Validation toolbar drop-down list.
- **Executable path** - Path to the executable file of the custom validation tool. You can insert here editor variables like ${home}, ${pd}, ${oxygenInstallDir}, etc.
- **Working directory** - The working directory of the custom validation tool.
- **Associated editors** - The editors which can perform validation with the external tool: the XML editor, the XSL editor, the XSD editor, etc.
- **Command line arguments for detected schemas** - Command line arguments used in the commands that validate the current edited file against different types of schema: W3C XML Schema, Relax NG full syntax, Relax NG compact syntax, NVDL, Schematron, DTD, etc.. The arguments can include any custom switch (like -rng) and the following editor variables:
  - ${cf} - Current file as file path, that is the absolute file path of the current edited document.
  - ${currentFileURL} - Current file as URL, that is the absolute file path of the current edited document represented as URL.
  - ${ds} - The path of the detected schema as a local file path for the current validated XML document.
  - ${dsu} - The path of the detected schema as an URL for the current validated XML document.

**Increasing the stack size for validation engines**

To prevent the appearance of a StackOverflowException, use one of the following methods:

- use the `com.oxygenxml.stack.size.validation.threads` property to increase the size of the stack for validation engines. The value of this property is specified in bytes. For example, to set a value of one megabyte specify 1x1024x1024=1048576;
- increase the value of the -Xss parameter.

**Note:** Increasing the value of the -Xss parameter affects all the threads of the application.
Fonts Preferences

Oxygen XML Developer plugin lets you choose the fonts used in the . To configure the Fonts options, open the Preferences dialog box and go to Fonts.

The following options are available:

- **Editor** - Sets the fonts used in the editor.
- **Note:** On Mac OS X, the default font, Monaco, cannot be rendered in bold.
- **Schema default font** - This option sets the font used in:
  - The Design mode of the XML Schema editor
  - Images with schema diagram fragments that are included in the HTML documentation generated from an XML Schema
- **Text** - This option sets the font used in Text mode. There are two options available:
  - **Map to text font** - Uses the same font as the one set in General / Appearance / Colors and Fonts for the basic text editor.
  - **Customize** - Allows you to choose a specific font.
- **Author** - This option sets the font to be used in Author mode.

Network Connection Settings Preferences

This section presents the options available in the Network Connection Settings preferences pages.

HTTP(S)/WebDAV Preferences

To set the HTTP(S)/WebDAV preferences, open the Preferences dialog box and go to Network Connection Settings > HTTP(S)/WebDAV. The following options are available:

- **Enable the HTTP(S)/WebDAV Protocols** - Activates the HTTP(S)/WebDAV protocols bundled with Oxygen XML Developer plugin. Any adjustment to this option requires a restart of the application.
- **Internal Apache HttpClient Version** - Oxygen XML Developer plugin uses the Apache HttpClient to establish connections to HTTP servers. To enable Oxygen XML Developer plugin to benefit from particular sets of features provided by different versions, you may choose between v3 and v4.
  - **Note:** For a full list of features, go to http://hc.apache.org/httpclient-3.x/ and http://hc.apache.org/httpcomponents-client-ga/

- **Maximum number of simultaneous connections per host** - Defines the maximum number of simultaneous connections established by the application with a distinct host. Servers might consider multiple connections opened from the same source to be a Denial of Service attack. You can avoid that by lowering the value of this option.
  - **Note:** This option accepts a minimum value of 5.

- **Read Timeout (seconds)** - The period in seconds after which the application considers that an HTTP server is unreachable if it does not receive any response to a request sent to that server.

- **Enable HTTP 'Expect: 100-continue' handshake for HTTP/1.1 protocol** - Activates Expect: 100-Continue handshake. The purpose of the Expect: 100-Continue handshake is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request headers) before the client sends the request body. The use of the Expect: 100-continue handshake can result in noticeable performance improvement when working with databases. The Expect: 100-continue handshake should be used with caution, as it may cause problems with HTTP servers and proxies that do not support the HTTP/1.1 protocol.
• **Use the 'Content-Type' header field to determine the content type** - When checked, tries to determine a resource type using the **Content-Type** header field. This header indicates the **Internet media type** of the message content, consisting of a type and subtype, for example:

```
Content-Type: text/xml
```

When unchecked, the resource type is determined by analyzing its extension. For example, a file ending in `.xml` is considered to be an XML file.

• **Automatic retry on recoverable error** - If enabled, if an HTTP error occurs when communicates with a server via HTTP, for example sending / receiving a SOAP request / response to / from a Web services server, and the error is recoverable, tries to send again the request to the server.

• **Automatically accept a security certificate, even if invalid** - When enabled, the HTTPS connections that Oxygen XML Developer plugin attempts to establish will accept all security certificates, even if they are invalid.

  ! **Important:** By accepting an invalid certificate, you accept at your own risk a potential security threat, since you cannot verify the integrity of the certificate’s issuer.

• **Encryption protocols (SVN Client only)** - this option is available only if you run the application with Java 1.6 or older. Sets a specific encryption protocol used when a repository is accessed through HTTPS protocol. You can choose one of the following values:

  • **SSLv3, TLSv1** (default value);
  • **SSLv3 only**;
  • **TLSv1 only**.

• **Lock WebDAV files on open** - If checked, the files opened through WebDAV are locked on the server so that they cannot be edited by other users while the lock placed by the current user still exists on the server.

(S)FTP Preferences

To configure the (S)FTP options, *open the Preferences dialog box* and go to **Network Connection Settings > (S)FTP**. You can customize the following options:

![Figure 245: The (S)FTP Configuration Preferences Panel](image)

• **Encoding for FTP control connection** - The encoding used to communicate with FTP servers: either ISO-8859-1 or UTF-8. If the server supports the UTF-8 encoding Oxygen XML Developer plugin will use it for communication. Otherwise it will use ISO-8859-1.

• **Public known hosts file** - File containing the list of all SSH server host keys that you have determined are accurate. The default value is `${homeDir}/.ssh/known_hosts`.

• **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol.

• **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol.

• **Show SFTP certificate warning dialog** - If checked, a warning dialog will be shown each time when the authenticity of the host cannot be established.
Scenarios Management Preferences

To configure Scenarios Management options, open the Preferences dialog box and go to Scenarios Management. This allows you to share the global transformation scenarios with other users by exporting them to an external file that can also be imported in this preferences panel.

![Scenarios Management](image)

**Figure 246: The Scenarios Management Preferences Panel**

The actions available in this panel are the following:

- **Import Global Transformation Scenarios** - Allows you to import at global level all transformation scenarios from a properties file created with the export scenario action. The names of the imported scenarios will appear in the Configure Transformation Scenario dialog followed by (import). This way there are no scenario name conflicts.

  If you want to work with project level scenarios you have to first switch to project level in the Configure Transformation Scenario dialog.

- **Export Global Transformation Scenarios** - Allows you to export all global transformation scenarios available in the Configure Transformation Scenario dialog.

- **Import Global Validation Scenarios** - Allows you to import at global level all scenarios from a properties file created with the export scenario action. The names of the imported scenarios will appear in the Configure Validation Scenario dialog followed by (import). This way there are no scenario name conflicts.

  If you want to work with project level scenarios you have to first switch to project level in the Configure Validation Scenario dialog.

- **Export Global Validation Scenarios** - Allows you to export all global validation scenarios available in the Configure Validation Scenario dialog.

View Preferences

To configure the view options, open the Preferences dialog box and go to Views and contains the following preferences:

- **Fixed width console** - If checked, a line in the Console view will be hard wrapped after the maximum numbers of characters allowed on a line.

- **Limit console output** - If checked, the content of the Console view will be limited to a configurable number of characters.

- **Console buffer** - Specifies the maximum number of characters that can be written in the Console view.

- **Tab width** - Specifies the number of spaces used for depicting a tab character.

XML Preferences

This section describes the panels that contain the user preferences related with XML.

**XML Catalog Preferences**

To configure the XML Catalog options, open the Preferences dialog box and go to XML > XML Catalog.

The following options are available:

- **Prefer** - the prefer setting determines whether public identifiers specified in the catalog are used in favor of system identifiers supplied in the document. Suppose you have an entity in your document for which both a public identifier and a system identifier has been specified, and the catalog only contains a mapping for the public identifier (e.g., a
matching public catalog entry). If **public** is selected, the URI supplied in the matching public catalog entry is used. If **system** is selected, the system identifier in the document is used.

**Note:** If the catalog contained a matching system catalog entry giving a mapping for the system identifier, that mapping would have been used, the public identifier would never have been considered, and the setting of override would have been irrelevant.

Generally, the purpose of catalogs is to override the system identifiers in XML documents, so **Prefer** should usually be **public** in your catalogs.

- When using catalogs it is sometimes useful to see what catalog files are parsed, if they are valid or not, and what identifiers are resolved by the catalogs. The **Verbosity** option selects the detail level of such logging messages of the XML catalog resolver that will be displayed in the **Catalogs** view at the bottom of the window:
  - **None** - No message is displayed by the catalog resolver when it tries to resolve a URI reference, a SYSTEM one or a PUBLIC one with the XML catalogs specified in this panel.
  - **Unresolved entities** - Only the logging messages that track the failed attempts to resolve references are displayed.
  - **All messages** - The messages of both failed attempts and successful ones are displayed.

- If **Resolve schema locations also through system mappings** is enabled, Oxygen XML Developer plugin analyzes both **uri** and **system** mappings in order to resolve the location of schema.

- If **Process namespaces through URI mappings for XML Schema** is selected then the target namespace of the imported XML Schemas is resolved through the **uri** mappings. The namespace is taken into account only when the schema specified in the **schemaLocation** attribute was not resolved successfully.

- If the **Use default catalog** option is checked the first XML catalog which Oxygen XML Developer plugin will use to resolve references at document validation and transformation will be a default built-in catalog. This catalog maps such references to the built-in local copies of the schemas of the Oxygen XML Developer plugin frameworks: DocBook, DITA, TEI, XHTML, SVG, etc.

You can also add or configure catalogs at framework level in the **Document Type Association** preferences page.

When you add, delete or edit an XML catalog to/from the list, reopen the currently edited files which use the modified catalog or run the **Validate action** so that the XML catalog changes take full effect.

**XML Parser Preferences**

To configure the **XML Parser** options, open the **Preferences dialog box** and go to **XML > XML Parser**.

The configurable options of the built-in XML parser are the following:

- **Enable parser caching (validation and content completion)** - enables re-use of internal models when validating and provides content completion in opened XML files which reference the same schemas (grammars) like DTD, XML Schema, RelaxNG.

- **Ignore the DTD for validation if a schema is specified** - forces validation against a referenced schema (W3C XML Schema, Relax NG schema) even if the document includes also a DTD DOCTYPE declaration. This option is useful when the DTD declaration is used only to declare DTD entities and the schema reference is used for validation against a W3C XML Schema or a Relax NG schema.

  **Note**: Schematron schemas are treated as additional schemas. The validation of a document associated with a DTD and referencing a Schematron schema is executed against both the DTD and the Schematron schema, regardless of the value of the **Ignore the DTD for validation if a schema is specified** option.

- **Enable XInclude processing** - enables XInclude processing. If checked, the XInclude support in Oxygen XML Developer plugin is turned on for validation and transformation of XML documents.

- **Base URI fix-up** - according to the specification for XInclude, processors must add an `xml:base` attribute to elements included from locations with a different base URI. Without these attributes, the resulting infoset information would be incorrect.
Unfortunately, these attributes make XInclude processing not transparent to Schema validation. One solution to this is to modify your schema to allow xml:base attributes to appear on elements that might be included from different base URIs.

If the addition of xml:base and /or xml:lang is undesired by your application, you can disable base URI fix-up.

- **Language fix-up** - the processor will preserve language information on a top-level included element by adding an xml:lang attribute if its include parent has a different [language] property. If the addition of xml:lang is undesired by your application, you can disable the language fix-up.
- **DTD post-validation** - enable this option to validate an XML file against the associated DTD, after all the content merged to the current XML file using XInclude was resolved. In case you disable this option, the current XML file is validated against the associated DTD before all the content merged to the current XML file using XInclude is resolved.

### XML Schema Preferences

To configure the XML Schema options, open the Preferences dialog box and go to XML > XML Parser > XML Schema.

This preferences page allows you to configure the following options:

- **Default XML Schema version** - Allows you to select the version of W3C XML Schema: XML Schema 1.0 or XML Schema 1.1.

  Note: You are also able to set the XML Schema version using the Customize option in New dialog box.

- **Default XML Schema validation engine** - Allows you to set the default XML Schema validation engine either to Xerces or Saxon EE.

### Xerces validation features

- **Enable full schema constraint checking** - Sets the schema-full-checking feature to true. This enables a validation of the parsed XML document against a schema (W3C XML Schema or DTD) while the document is parsed.

- **Enable honour all schema location feature** - Sets the honour-all-schema-location feature to true. All the files that declare W3C XML Schema components from the same namespace are used to compose the validation model. In case this option is disabled, only the first W3C XML Schema file that is encountered in the XML Schema import tree is taken into account.

- **Enable full XPath 2.0 in assertions and alternative types** - When enabled (default value), you can use the full XPath support in assertions and alternative types. Otherwise, only the XPath support offered by the Xerces engine is available.

- **Assertions can see comments and processing instructions** - Controls whether comments and processing instructions are visible to the XPath expression used for defining an assertion in XSD 1.1.

### Saxon validation features

- **Multiple schema imports** - Forces xs:import to fetch the referenced schema document. By default, the xs:import fetches the document only if no schema document for the given namespace has already been loaded. With this option in effect, the referenced schema document is loaded unless the absolute URI is the same as a schema document already loaded.

- ** Assertions can see comments and processing instructions** - Controls whether comments and processing instructions are visible to the XPath expression used to define an assertion. By default (unlike Saxon 9.3), they are not made visible.

### Relax NG Preferences

To configure the Relax NG options, open the Preferences dialog box and go to XML > XML Parser > Relax NG.

The following options are available in this page:
• **Check feasibly valid** - Checks whether Relax NG documents can be transformed into valid documents by inserting any number of attributes and child elements anywhere in the tree.

  Note: Enabling this option disables the **Check ID/IDREF** option.

• **Check ID/IDREF** - Checks the ID/IDREF matches when a Relax NG document is validated.

• **Add default attribute values** - Default values are given to the attributes of documents validated using Relax NG. These values are defined in the Relax NG schema.

**Schematron Preferences**

To configure the Schematron options, *open the Preferences dialog box* and go to XML > XML Parser > Schematron.

The following options are available in this preferences page:

• **Schematron XPath Version** - selects the version of XPath for the expressions that are allowed in Schematron assertion tests: 1.0 or 2.0. This option is applied both in standalone Schematron schemas and in embedded Schematron rules, both in Schematron 1.5 and in ISO Schematron.

• **Optimize (visit-no-attributes)** - in case your ISO Schematron assertion tests do not contain the attributes axis you should check this option for faster ISO Schematron validation.

• **Allow foreign elements (allow-foreign)** - enables support for allow-foreign on ISO Schematron. This option is used to pass non-Schematron elements to the generated stylesheet.

• **Use Saxon EE (schema aware) for xslt2 query binding** - when enabled, Saxon EE is used for xslt2 query binding. In case this option is disabled, Saxon PE is used.

• **Enable Schematron Quick Fixes (SQF) support** - Allows you to enable or disable the support for quick fixes in Schematron files. This option is enabled by default.

**XML Instances Generator Preferences**

To configure the XML Instances Generator options, *open the Preferences dialog box* and go to XML > XML Instances Generator. It sets the default parameters of the Generate Sample XML Files tool that is available on the Tools menu.

The options of the tool that generates XML instance documents based on a W3C XML Schema are the following:

• **Generate optional elements** - If checked, the elements declared optional in the schema will be generated in the XML instance.

• **Generate optional attributes** - If checked, the attributes declared optional in the schema will be generated in the XML instance.

• **Values of elements and attributes** - Specifies what values are generated in elements and attributes of the XML instance. It can have one of the values:
  • **None** - no values for the generated elements and attributes
  • **Default** - the value is the element name or attribute name
  • **Random** - a random value

• **Preferred number of repetitions** - If the values set here is greater than maxOccurs, then the maxOccurs is used.

• **Maximum recursivity level** - For recursive type definitions this parameter specifies the number of levels of recursive elements inserted in the parent element with the same name.

• **Type alternative strategy** - Specifies how the element type alternatives are generated in the XML instance:
  • **First** - the first element type alternative whose XPath condition is true is used;
  • **Random** - a random element type alternative whose XPath condition is true is used;

  Note: In case no XPath condition is true, the default element type alternative is used.

• **Choice strategy** - For choice element models specifies what choice will be generated in the XML instance:
• **First** - the first choice is selected from the choice definition and an instance of that choice is generated in the XML instance document.

• **Random** - a random choice is selected from the choice definition and an instance of that will be generated.

• **Generate the other options as comments** - If checked, the other options of the choice element model (the options which are not selected) will be generated inside an XML comment in the XML instance.

• **Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an a element the generated values are: a1, a2, a3, etc. If not checked, the value is the name of the type of that element / attribute, for example: string, decimal, etc.

• **Maximum length** - The maximum length of string values generated for elements and attributes.

• **Discard optional elements after nested level** - The optional elements that exceed the specified nested level are discarded. This option is useful for limiting deeply nested element definitions that can quickly result in very large XML documents.

**XProc Engines Preferences**

Oxygen XML Developer plugin comes with a built-in XProc engine called Calabash. An external XProc engine can be configured in this panel.

When **Show XProc messages** is selected all messages emitted by the XProc processor during a transformation will be presented in the results view.

For an external engine the value of the **Name** field will be displayed in the XProc transformation scenario and in the command line that will start it.

![Command line](image)

**Figure 247: Creating an XProc external engine**

Other parameters that can be set for an XProc external engine are the following: , and the error stream of the engine, the working directory of the command that will start the engine. The encodings will be used for reading and displaying the output of the engine. The working directory and

• a textual description that will appear as tooltip where the XProc engine will be used
• the encoding for the output stream of the XProc engine, used for reading and displaying the output messages
• the encoding for the error stream of the XProc engine, used for reading and displaying the messages from the error stream
• the working directory for resolving relative paths
• the command line that will run the XProc engine as an external process; the command line can use built-in editor variables and custom editor variables for parametrizing a file path

Note: You can configure the Saxon processor using the saxon.config file. For further details about configuring this file go to http://www.saxonica.com/documentation9.5/index.html#!configuration/configuration-file.

Note: You can configure Calabash using the calabash.config file.

Note: These files are located in [OXYGEN_DIR]\lib\xproc\calabash. In case they do not exist, you have to create them.

XSLT-FO-XQuery Preferences

To configure the XSLT/FO/XQuery options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery. This panel contains only the most generic options for working with XSLT / XSL-FO / XQuery processors. The more specific options are grouped in other panels linked as child nodes of this panel in the tree of the Preferences dialog.

There is only one generic option available:

Create transformation temporary files in system temporary directory - It should be selected only when the temporary files necessary for performing transformations are created in the same folder as the source of the transformation (the default behavior, when this option is not selected) and this breaks the transformation. An example of breaking the transformation is when the transformation processes all the files located in the same folder as the source of the transformation, which will include the temporary files, and the result is incorrect or the transformation fails due to this fact.

XSLT Preferences

To configure the XSLT options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XSLT.

Oxygen XML Developer plugin gives you the possibility to use an XSLT transformer implemented in Java (other than the XSLT transformers that come bundled with Oxygen XML Developer plugin). To use a different XSLT transformer, specify the name of the transformer factory class. Oxygen XML Developer plugin sets this transformer factory class as the value of the Java property javax.xml.transform.TransformerFactory.

You can customize the following XSLT preferences:

• Value - Allows the user to enter the name of the transformer factory Java class.
• XSLT 1.0 Validate with - allows you to set the XSLT engine used for validation of XSLT 1.0 documents.
• XSLT 2.0 Validate with - allows you to set the XSLT Engine used for validation of XSLT 2.0 documents.
• XSLT 3.0 Validate with - allows you to set the XSLT Engine used for validation of XSLT 3.0 documents.

Note: Saxon-HE does not implement any XSLT 3.0 features. Saxon-PE implements a selection of XSLT 3.0 (and XPath 3.0) features, with the exception of schema-awareness and streaming. Saxon-EE implements additional features relating to streaming (processing of a source document without constructing a tree in memory. For further details about XSLT 3.0 conformance, go to http://www.saxonica.com/documentation/index.html#!conformance/xslt30.

Saxon6 Preferences

To configure the Saxon 6 options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon 6.
The built-in Saxon 6 XSLT processor can be configured with the following options:

- **Line numbering** - Specifies whether line numbers are maintained and reported in error messages for the XML source document.
- **Disable calls on extension functions** - If enabled, external functions called is disallowed. Checking this is recommended in an environment where untrusted stylesheets may be executed. Also disables user-defined extension elements, together with the writing of multiple output files, all of which carry similar security risks.
- **Handling of recoverable stylesheet errors** - Allows the user to choose how dynamic errors are handled. Either one of the following options can be selected:
  - recover silently - Continue processing without reporting the error.
  - recover with warnings - Issue a warning but continue processing.
  - signal the error and do not attempt recovery - Issue an error and stop processing.

**Saxon-HE/PE/EE Preferences**

To configure the Saxon HE/PE/EE options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE.

Oxygen XML Developer plugin allows you to configure the following XSLT options for the Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE):

- **Use a configuration file** ("-config") - Sets a Saxon 9.6.0.5 configuration file that is executed for XSLT transformation and validation processes.
- **Version warnings** ("-versmsg") - Warns you when the transformation is applied to an XSLT 1.0 stylesheet.
- **Line numbering** ("-l") - Error line numbers are included in the output messages.
- **Debugger trace into XPath expressions (applies to debugging sessions)** - Instructs the XSLT Debugger to step into XPath expressions.
- **Expand attributes defaults** ("-expand") - Specifies whether or not the attributes defined in the associated DTD or XML Schema are expanded in the output of the transformation you are executing.
- **DTD validation of the source** ("-dtd") - The following options are available:
  - **On** - Requests DTD validation of the source file and of any files read using the `document()` function.
  - **Off** - (default setting) Suppresses DTD validation.
  - **Recover** - Performs DTD validation but treats the errors as non-fatal.

  Note: Any external DTD is likely to be read even if not used for validation, since DTDs can contain definitions of entities.

- **Recoverable errors** ("-warnings") - Policy for handling recoverable errors in the stylesheet: Allows you to choose how dynamic errors are handled. One of the following options can be selected:
  - **Recover silently** ("silent") - Continues processing without reporting the error.
  - **Recover with warnings** ("recover") - (default setting) Issues a warning but continues processing.
  - **Signal the error and do not attempt recovery** ("fatal") - Issues an error and stops processing.

- **Strip whitespaces** ("-strip") - Strip whitespaces feature can be one of the following three options:
All ("all") - Strips all whitespace text nodes from source documents before any further processing, regardless of any xsl:strip-space declarations in the stylesheet, or any xml:space attributes in the source document.

Ignorable ("ignorable") - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any xsl:strip-space declarations in the stylesheet, or any xml:space attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.

None ("none") - default setting. No whitespaces are stripped before further processing. However, whitespace are stripped if this is specified in the stylesheet using xsl:strip-space.

Optimization level ("-opt") - Set the optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization). This option allows optimization to be suppressed in cases where reducing the compiling time is important, where optimization conflicts with debugging, or causes extension functions with side-effects to behave unpredictably.

The advanced options available only in Saxon Professional Edition (PE) and Enterprise Edition (EE) are:

Allow calls on extension functions ("-ext") - If checked, the stylesheet is allowed to call external Java functions. This does not affect calls on integrated extension functions, including Saxon and EXSLT extension functions. This option is useful when loading an untrusted stylesheet, such as from a remote site using an http://[URL]. It ensures that the stylesheet cannot call arbitrary Java methods and thus gain privileged access to resources on your machine.

Register Saxon-CE extension functions and instructions - Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.6.0.5 processors.

Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about executing such a stylesheet on Saxonica's website.

The advanced options available only in Saxon Enterprise Edition (EE) are:

XML Schema version - Use this option to change the default XML Schema version. To change the default XML Schema version, open the Preferences dialog box and go to XML > XML Parser > XML Schema.

Note: This option is available when you configure the Saxon EE advanced options from a transformation scenario.

Validation of the source file ("-val") - Requests schema-based validation of the source file and of any files read using the document() or similar functions. It can have the following values:

Schema validation ("strict") - This mode requires an XML Schema and specifies that the source documents should be parsed with strict schema-validation enabled.

Lax schema validation ("lax") - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.

Disable schema validation - This specifies that the source documents should be parsed with schema-validation disabled.

Validation errors in the results tree treated as warnings ("-outval") - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.

Write comments for non-fatal validation errors of the result document - The validation messages are written (where possible) as a comment in the result document itself.

Generate bytecode ("--generateByteCode:(on|off)") - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to http://www.saxonica.com/documentation9.5/index.html#javadoc.

Saxon HE/PE/EE Advanced Preferences

To configure the Saxon HE/PE/EE Advanced preferences, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE > Advanced.

- **URI Resolver class name ("-r")** - Specifies a custom implementation for the URI resolver used by the XSLT Saxon 9.6.0.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from the dialog for configuring the XSLT extension for the particular transformation scenario.

- **Collection URI Resolver class name ("-cr")** - Specifies a custom implementation for the Collection URI resolver used by the XSLT Saxon 9.6.0.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from the dialog for configuring the XSLT extension for the particular transformation scenario.

**XSLTProc Preferences**

To configure XSLTProc options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XSLT > XSLTProc.

The options of the XSLTProc processor are the same as the ones available in the command line:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when XSLTProc is used as transformer in XSLT transformation scenarios.

- **Skip loading the document’s DTD** - If checked, the DTD specified in the DOCTYPE declaration will not be loaded.

- **Do not apply default attributes from document’s DTD** - If checked, the default attributes declared in the DTD and not specified in the document are not included in the transformed document.

- **Do not use Internet to fetch DTD’s, entities or docs** - If checked, the remote references to DTD’s and entities are not followed.

- **Maximum depth in templates stack** - If this limit of maximum templates depth is reached the transformation ends with an error.

- **Verbosity** - If checked, the transformation will output detailed status messages about the transformation process in the Warnings view.

- **Show version of libxml and libxslt used** - If checked, Oxygen XML Developer plugin will display in the Warnings view the version of the libxml and libxslt libraries invoked by XSLTProc.

- **Show time information** - If checked, the Warnings view will display the time necessary for running the transformation.

- **Show debug information** - If checked, the Warnings view will display debug information about what templates are matched, parameter values, etc.

- **Show all documents loaded during processing** - If checked, Oxygen XML Developer plugin will display in the Warnings view the URL of all the files loaded during transformation.

- **Show profile information** - If checked, Oxygen XML Developer plugin will display in the Warnings view a table with all the matched templates, and for each template will display: the match XPath expression, the template name, the number of template modes, the number of calls, the execution time.

- **Show the list of registered extensions** - If checked, Oxygen XML Developer plugin will display in the Warnings view a list with all the registered extension functions, extension elements and extension modules.

- **Refuses to write to any file or resource** - If checked, the XSLTProc processor will not write any part of the transformation result to an external file on disk. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.
• **Refuses to create directories** - If checked, the XSLTProc processor will not create any directory during the transformation process. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

**MSXML Preferences**

To configure the MSXML options, open the Preferences dialog box and go to **XML > XSLT/FO/XQuery > XSLT > MSXML**.

The options of the MSXML 3.0 and 4.0 processors are the same as the ones available in the command line for the MSXML processors:

- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.

- **Do not resolve external definitions during parse phase** - By default, MSXSL instructs the parser to resolve external definitions such as document type definition (DTD), external subsets or external entity references when parsing the source and style sheet documents. If this option is checked the resolution is disabled.

- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.

- **Show time information** - If checked, the relative speed of various transformation steps can be measured:
  - The time to load, parse, and build the input document.
  - The time to load, parse, and build the stylesheet document.
  - The time to compile the stylesheet in preparation for the transformation.
  - The time to execute the stylesheet.

- **Start transformation in this mode** - Although stylesheet execution usually begins in the empty mode, this default may be changed by specifying another mode. Changing the start mode allows execution to jump directly to an alternate group of templates.

**MSXML.NET Preferences**

To configure the MSXML.NET options, open the Preferences dialog box and go to **XML > XSLT/FO/XQuery > XSLT > MSXML.NET**.

The options of the MSXML.NET processor are:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when MSXML.NET is used as transformer in the XSLT transformation scenario.

- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.

- **Do not resolve external definitions during parse phase** - By default MSXML.NET resolves external definitions such as DTD external subsets or external entity references when parsing source XML document and stylesheet document. Using this option you can disable this behaviour. Note, that it may affect also the validation process for the XML document.

- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.

- **Show time information** - If checked, the relative speed of various transformation steps can be measured:
  - The time to load, parse, and build the input document.
  - The time to load, parse, and build the stylesheet document.
  - The time to compile the stylesheet in preparation for the transformation.
  - The time to execute the stylesheet.

- **Forces ASCII output encoding** - There is a known problem with .NET 1.X XSLT processor (System.Xml.Xsl.XslTransform class): it doesn't support escaping of characters as XML character references when they cannot be represented in the output encoding. That means that when you output a character that cannot be represented in output encoding, it will be outputted as '?'. Usually this happens when output encoding is set to
ASCII. With this option checked the output is forced to be ASCII encoded and all non-ASCII characters get escaped as XML character references (&#nnnn; form).

- **Allow multiple output documents** - This option allows to create multiple result documents using the `exsl:document` extension element.

- **Use named URI resolver class** - This option allows to specify a custom URI resolver class to resolve URI references in `xsl:import` and `xsl:include` instructions (during XSLT stylesheet loading phase) and in `document()` function (during XSL transformation phase).

- **Assembly file name for URI resolver class** - The previous option specifies partially or fully qualified URI resolver class name, e.g. `Acme.Resolvers.CacheResolver`. Such name requires additional assembly specification using this option or the next option, but fully qualified class name (which always includes an assembly specifier) is all-sufficient. See MSDN for more info about *fully qualified class names*. This option specifies a file name of the assembly, where the specified resolver class can be found.

- **Assembly GAC name for URI resolver class** - This option specifies partially or fully qualified name of the assembly in the *global assembly cache* (GAC), where the specified resolver class can be found. See MSDN for more info about *partial assembly names*. Also see the previous option.

- **List of extension object class names** - This option allows to specify extension object classes, whose public methods then can be used as extension functions in an XSLT stylesheet. It is a comma-separated list of namespace-qualified extension object class names. Each class name must be bound to a namespace URI using prefixes, similar to providing XSLT parameters.

- **Use specified EXSLT assembly** - MSXML.NET supports a rich library of the EXSLT and EXSLT.NET extension functions embedded or in a plugged in EXSLT.NET library. EXSLT support is enabled by default and cannot be disabled in this version. If you want to use an external EXSLT.NET implementation instead of a built-in one use this option.

- **Credential loading source xml** - This option allows to specify user credentials to be used when loading XML source documents. The credentials should be provided in the `username:password@domain` format (all parts are optional).

- **Credential loading stylesheet** - This option allows to specify user credentials to be used when loading XSLT stylesheet documents. The credentials should be provided in the `username:password@domain` format (all parts are optional).

**XQuery Preferences**

To configure the XQuery options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XQuery.

The generic XQuery preferences are the following:

- **XQuery validate with** - Allows you to select the processor that validates XQuery documents. In case you are validating an XQuery file that has an associated validation scenario, Oxygen XML Developer plugin uses the processor specified in the scenario. If no validation scenario is associated, but the file has an associated transformation scenario, the processor specified in the scenario is used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor.

- **Size limit of Sequence view (MB)** - When the result of an XQuery transformation is *set in the transformation scenario as sequence* the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceed, go to the **Sequence** view and click **More results available** to extract more data from the database.

- **Format transformer output** - Specifies whether the output of the transformer is formatted and indented (pretty printed).

  **Note:** This option is ignored if you choose **Sequence** (lazy extract data from a database) from the associated transformation scenario.

- **Create structure indicating the type nodes** - If checked, Oxygen XML Developer plugin takes the results of a query and creates an XML document containing copies of all items in the sequence, suitably wrapped.

  **Note:** This option is ignored if you choose **Sequence** (lazy extract data from a database) from the associated transformation scenario.
Saxon HE/PE/EE Preferences

To configure the Saxon HE/PE/EE options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE.

The XQuery preferences for the Saxon 9.6.0.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that is used for XQuery transformation and validation.

- **Recoverable errors ("-warnings")** - Allows the user to choose how dynamic errors are handled. The following options can be selected:
  - **recover silently ("silent")** - Continues processing without reporting the error.
  - **recover with warnings ("recover")** - Issues a warning but continues processing.
  - **signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.

- **Strip whitespaces ("-strip")** - Can have one of the following values:
  - **All ("all")** - Strips all whitespace text nodes from source documents before any further processing, regardless of any xml:space attributes in the source document.
  - **Ignore ("ignoreable")** - Strips all ignoreable whitespace text nodes from source documents before any further processing, regardless of any xml:space attributes in the source document. Whitespace text nodes are ignoreable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None ("none")** - Strips no whitespace before further processing.

- **Optimization level ("-opt")** - Set the optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization). This option allows optimization to be suppressed in cases where reducing the compiling time is important, where optimization conflicts with debugging, or causes extension functions with side-effects to behave unpredictably.

- **Use linked tree model ("-tree:linked")** - This option activates the linked tree model.

- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - If checked, Saxon runs the XQuery transformation with the XQuery 3.0 support (this option is enabled by default).

The following option is available for Saxon 9.6.0.5 Professional Edition (PE) and Enterprise Edition (EE) only:

- **Allow calls on extension functions ("-ext")** - If checked, calls on external functions are allowed. Checking this option is recommended in an environment where untrusted stylesheets may be executed. It also disables user-defined extension elements and the writing of multiple output files, both of which carry similar security risks.

The options available specifically for Saxon 9.6.0.5 Enterprise Edition (EE) are as follows:

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and enables parsing the source documents with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.

- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.

- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.

- **Generate bytecode ("--generateByteCode:(on|off)")** - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to [http://www.saxonica.com/documentation9.5/index.html#javadoc](http://www.saxonica.com/documentation9.5/index.html#javadoc).

- **Enable XQuery update ("-update:(on|off)")** - This option controls whether or not XQuery update syntax is accepted.
• **Backup files updated by XQuery** ("-backup:(on|off)") - If checked, backup versions for any XML files updated with XQuery Update are generated. This option is available when the Enable XQuery update option is enabled.

**Saxon HE/PE/EE Advanced Preferences**

To configure Saxon HE/PE/EE Advanced preferences, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE > Advanced.

![Advanced Preferences Panel](image)

**Figure 250:** The Saxon HE/PE/EE XQuery Advanced Preferences Panel

The advanced XQuery options which can be configured for the Saxon 9.6.0.5 XQuery transformer (all editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **URI Resolver class name** - Allows you to specify a custom implementation for the URI resolver used by the XQuery Saxon 9.6.0.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from the dialog for configuring the XQuery extension for the particular transformation scenario.

  **Note:** If your URIResolver implementation does not recognize the given resource, the `resolve(String href, String base)` method should return a `null` value. Otherwise, the given resource will not be resolved through the XML catalog.

- **Collection URI Resolver class name** - Allows you to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.6.0.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from the dialog for configuring the XQuery extension for the particular transformation scenario.

**Debugger Preferences**

To configure the Debugger preferences, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > Debugger.

The following preferences are available:

- **Show xsl:result-document output** - if checked, the debugger presents the output of `xsl:result-document` instructions into the debugger output view;

- **Infinite loop detection** - set this option to receive notifications when an infinite loop occurs during transformation;

- **Maximum depth in templates stack** - sets how many `xsl:template` instructions can appear on the current stack. This setting is used by the infinite loop detection;

- **Debugger layout** - a horizontal layout means that the stack of XML editors takes the left half of the editing area and the stack of XSL editors takes the right one. A vertical layout means that the stack of XML editors takes the upper half of the editing area and the stack of XSL editors takes the lower one;

- **XWatch evaluation timeout (seconds)** - specifies the maximum time that Oxygen XML Developer plugin allocates to the evaluation of XPath expressions while debugging;

- **Messages** - specifies whether a debugging session is stopped, is continued, or you are asked what to do when you are editing the source document involved in a debugging session.

**Annotations Preferences**

To configure the Annotations options, go to Window (Eclipse on Mac OSX) and choose Preferences. Then go to General > Editors > Text Editors > Annotations.

The following Oxygen XML Developer plugin preferences are available:
• **XSLT/XQuery Debug Current Instruction Pointer** - Controls the background color of the current execution node, both in the document (XML) and XSL/XQuery views.

**Profiler Preferences**

This section explains the settings available for the XSLT Profiler. To access and modify these settings, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > Profiler (see Debugger Preferences on page 535).

The following profiles settings are available:

• **Show time** - Shows the total time that was spent in the call.
• **Show inherent time** - Shows the inherent time that was spent in the call. The inherent time is defined as the total time of a call minus the time of its child calls.
• **Show invocation count** - Shows how many times the call was called in this particular call sequence.
• **Time scale** - The time scale options determine the unit of time measurement, which may be milliseconds or microseconds.
• **Hotspot threshold** - The threshold below which hot spots are ignored (milliseconds).
• **Ignore invocation less than** - The threshold below which invocations are ignored (microseconds).
• **Percentage calculation** - The percentage base determines against what time span percentages are calculated:
  - **Absolute** - Percentage values show the contribution to the total time.
  - **Relative** - Percentage values show the contribution to the calling call.

**FO Processors Preferences**

Besides Apache FOP, the built-in formatting objects processor, you can configure other external processors and set them in the transformation scenarios for processing XSL-FO documents.

Oxygen XML Developer plugin provides an easy way to add two of the most used commercial FO processors: RenderX XEP and Antenna House XSL Formatter. You can easily add RenderX XEP as an external FO processor if you have the XEP installed. Also, if you have the Antenna House XSL Formatter v4 or v5, Oxygen XML Developer plugin uses the environmental variables set by the XSL Formatter installation to detect and use it for XSL-FO transformations. If the environmental variables are not set for the XSL Formatter installation, you can browse and choose the executable file just as you would for XEP. You can use these two external FO processors for DITA OT transformations scenarios and XML with XSLT transformation scenarios.

To configure the FO Processors options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > FO Processors.
The options for FO processors are the following:

- **Use built-in Apache FOP** - instructs Oxygen XML Developer plugin to use its built-in Apache FO processor.
- **Use other Apache FOP** - instructs Oxygen XML Developer plugin to use another Apache FO processor installed on your computer.
- **Enable the output of the built-in FOP** - all Apache FOP output is displayed in a results pane at the bottom of the Oxygen XML Developer plugin window including warning messages about FO instructions not supported by Apache FOP.
- **Memory available to the built-in FOP** - if your Apache FOP transformations fail with an Out of Memory error (OutOfMemoryError) select from this combo box a larger value for the amount of memory reserved for FOP transformations.
- **Configuration file for the built-in FOP** - you should specify here the path to an Apache FOP configuration file, necessary for example to render to PDF a document containing Unicode content using a special true type font.
- **Generates PDF/A-1b output** - when selected PDF/A-1b output is generated.

**Note:** All fonts have to be embedded, even the implicit ones. More information about configuring metrics files for the embedded fonts can be found in *Add a font to the built-in FOP*.

**Note:** You cannot use the `<filterList>` key in the configuration file because FOP would generate the following error: *The Filter key is prohibited when PDF/A-1 is active.*

- **Add 'XEP' FO processor (executable file is needed)** - in case RenderX XEP is already installed on your computer, you can use this button to choose the XEP executable script (xep.bat for Windows, xep for Linux).
- **Add 'Antenna House' FO processor (executable file is needed)** - in case Antenna House XSL Formatter is already installed on your computer, you can use this button to choose the Antenna House executable script (AHFCmd.exe, or XSLCmd.exe for Windows, AHFCmd.sh, or XSLCmd.sh for Linux).
External FO processors

In this section you can manage the external FO processors you want to use in transformation scenarios. Press the New button to add a new external FO processor. The following dialog is displayed:

![Figure 252: The External FO Processor Configuration Dialog](image)

- **Name** - the name displayed in the list of available FOP processors on the FOP tab of the transformation scenario dialog.
- **Description** - a textual description of the FO processor displayed in the FO processors table and in tooltips of UI components where the processor is selected.
- **Working directory** - the directory where the intermediate and final results of the processing is stored. Here you can use one of the following editor variables:
  - `${homeDir}` - the path to user home directory.
  - `${cfd}` - the path of current file directory. If the current file is not a local file, the target is the user's desktop directory.
  - `${pd}` - the project directory.
  - `${oxygenInstallDir}` - the Oxygen XML Developer plugin installation directory.
- **Command line** - the command line that starts the FO processor, specific to each processor. Here you can use one of the following editor variables:
  - `${method}` - the FOP transformation method: pdf, ps or txt
  - `${fo}` - the input FO file
  - `${out}` - the output file
  - `${pd}` - the project directory
  - `${frameworksDir}` - the path of the frameworks subdirectory of the Oxygen XML Developer plugin install directory
  - `${oxygenInstallDir}` - the Oxygen XML Developer plugin installation directory
  - `${ps}` - the platform-specific path separator. It is used between the library files specified in the class path of the command line
- **Output Encoding** - the encoding of the FO processor output stream displayed in a results panel at the bottom of the Oxygen XML Developer plugin window.
- **Error Encoding** - the encoding of the FO processor error stream displayed in a results panel at the bottom of the Oxygen XML Developer plugin window.
XPath Preferences

To configure the XPath options, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > XPath.

Oxygen XML Developer plugin allows you to customize the following options:

- **Unescape XPath expression** - the entities of an XPath expressions that you type in the XPath/XQuery Builder are unescaped during their execution. For example the expression

  ```xml
  //varlistentry[starts-with(@os,'s')]
  ```

  is equivalent with:

  ```xml
  //varlistentry[starts-with(@os,'s')]
  ```

- **No namespace** - if you enable this option, Oxygen XML Developer plugin considers unprefixed element names of the evaluated XPath 2.0 / 3.0 expressions as belonging to no namespace.

- **Use the default namespace from the root element** - if you enable this option, Oxygen XML Developer plugin considers unprefixed element names of the evaluated XPath expressions as belonging to the default namespace declared on the root element of the XML document you are querying.

- **Use the namespace of the root** - if you enable this option, Oxygen XML Developer plugin considers unprefixed element names of the evaluated XPath expressions as belonging to the same namespace as the root element of the XML document you are querying.

- **This namespace** - in this field you can enter the namespace of the unprefixed elements.

- **Default prefix-namespace mappings** - in this field you can associate prefixes with namespaces. Use these mappings when you want to define them globally, not for each document.

Custom Engines Preferences

You can configure and run XSLT and XQuery transformations with processors other than the ones which come with the Oxygen XML Developer plugin distribution.

Note:

To configure the Custom Engines preferences, open the Preferences dialog box and go to XML > XSLT/FO/XQuery > Custom Engines.

The following parameters can be configured for a custom engine:

![Figure 253: Parameters of a Custom Engine](image)

Figure 253: Parameters of a Custom Engine
• **Engine type** - Combo box allowing you to choose the transformer type. There are two options: XSLT engines and XQuery engines.

• **Name** - The name of the transformer displayed in the dialog for editing transformation scenarios

• **Description** - A textual description of the transformer.

• **Working directory** - The start directory of the transformer executable program. The following editor variables are available for making the path to the working directory independent of the location of the input files:
  * `${homeDir}` - The user home directory in the operating system.
  * `${cfd}` - The path to the directory of the current file.
  * `${pd}` - The path to the directory of the current project.
  * `${oxygenInstallDir}` - The Oxygen XML Developer plugin install directory.

• **Command line** - The command line that must be executed by Oxygen XML Developer plugin to perform a transformation with the engine. The following editor variables are available for making the parameters in the command line (the transformer executable, the input files) independent of the location of the input files:
  * `${xml}` - The XML input document as a file path.
  * `${xmlu}` - The XML input document as a URL.
  * `${xsl}` - The XSL / XQuery input document as a file path.
  * `${xslu}` - The XSL / XQuery input document as a URL.
  * `${out}` - The output document as a file path.
  * `${outu}` - The output document as a URL.
  * `${ps}` - The platform separator which is used between library file names specified in the class path.

• **Output Encoding** - The encoding of the transformer output stream.

• **Error Encoding** - The encoding of the transformer error stream.

**Import Preferences**

To configure the **Import** options, open the **Preferences dialog box** and go to **XML > Import**. This page allows you to configure how empty values and **null** values are handled when they are encountered in imported database tables or Excel sheets. Also you can configure the format of date / time values recognized in the imported database tables or Excel sheets.

The following options are available:

• **Create empty elements for empty values** - If checked, an empty value from a database column or from a text file is imported as an empty element.

• **Create empty elements for null values** - If checked, **null** values from a database column are imported as empty elements.

• **Escape XML content** - Enabled by default, this option instructs Oxygen XML Developer plugin to escape the imported content to an XML-safe form.

• **Add annotations for generated XML Schema** - If checked, the generated XML Schema contains an annotation for each of the imported table columns. The documentation inside the annotation tag contains the remarks of the database columns (if available) and also information about the conversion between the column type and the generated XML Schema type.

The section **Date / Time Format** specifies the format used for importing date and time values from Excel spreadsheets or database tables and in the generated XML schemas. The following format types are available:

• **Unformatted** - If checked, the date and time formats specific to the database are used for import. When importing data from Excel a string representation of date or time values are used. The type used in the generated XML Schema is `xs:string`.

• **XML Schema date format** - If checked, the XML Schema-specific format ISO8601 is used for imported date / time data (`yyyy-MM-dd 'T' HH:mm:ss` for datetime, `yyyy-MM-dd` for date and `HH:mm:ss` for time). The types used in the generated XML Schema are `xs:datetime`, `xs:date` and `xs:time`. 


**Custom format** - If checked, the user can define a custom format for timestamp, date, and time values or choose one of the predefined formats. A preview of the values is presented when a format is used. The type used in the generated XML Schema is `xs:string`.

### Date / Time Patterns Preferences

**Table 11: Pattern letters**

<table>
<thead>
<tr>
<th>Letter</th>
<th>Date or Time Component</th>
<th>Presentation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era designator</td>
<td>Text</td>
<td>AD</td>
</tr>
<tr>
<td>y</td>
<td>Year</td>
<td>Year</td>
<td>1996; 96</td>
</tr>
<tr>
<td>M</td>
<td>Month in year</td>
<td>Month</td>
<td>July; Jul; 07</td>
</tr>
<tr>
<td>w</td>
<td>Week in year</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>W</td>
<td>Week in month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>Day in year</td>
<td>Number</td>
<td>189</td>
</tr>
<tr>
<td>d</td>
<td>Day in month</td>
<td>Number</td>
<td>10</td>
</tr>
<tr>
<td>F</td>
<td>Day of week in month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>Day in week</td>
<td>Text</td>
<td>Tuesday; Tue</td>
</tr>
<tr>
<td>a</td>
<td>Am / pm marker</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>H</td>
<td>Hour in day (0-23)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>k</td>
<td>Hour in day (1-24)</td>
<td>Number</td>
<td>24</td>
</tr>
<tr>
<td>K</td>
<td>Hour in am / pm (0-11)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>Hour in am / pm (1-12)</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>m</td>
<td>Minute in hour</td>
<td>Number</td>
<td>30</td>
</tr>
<tr>
<td>s</td>
<td>Second in minute</td>
<td>Number</td>
<td>55</td>
</tr>
<tr>
<td>S</td>
<td>Millisecond</td>
<td>Number</td>
<td>978</td>
</tr>
<tr>
<td>z</td>
<td>Time zone</td>
<td>General time zone</td>
<td>Pacific Standard Time; PST; GMT-08:00</td>
</tr>
<tr>
<td>Z</td>
<td>Time zone</td>
<td>RFC 822 time zone</td>
<td>-0800</td>
</tr>
</tbody>
</table>

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:
  - **`GMTsOffsetTimeZone`** - GMT Sign Hours : Minutes
  - **`Sign`** - one of + or -
  - **`Hours`** - one of or two digits
  - **`Minutes`** - two digits
  - **`Digit`** - one of 0 1 2 3 4 5 6 7 8 9
Hours must be between 0 and 23, and Minutes must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

- **RFC 822 time zone**: The RFC 822 4-digit time zone format is used:
  - `RFC822TimeZone` - Sign `TwoDigitHours` Minutes
  - `TwoDigitHours` - a number of two digits

  TwoDigitHours must be between 00 and 23.

**XML Signing Certificates Preferences**

Oxygen XML Developer plugin provides two types of **keystores** for certificates that are used for digital signatures of XML documents: Java KeyStore (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A keystore file is protected by a password. To configure a certificate keystore, open the Preferences dialog box and go to XML > XML Signing Certificates. You can customize the following parameters of a keystore:

![Certificates for Signing XML Documents](image)

**Figure 254: The Certificates Preferences Panel**

- **Keystore type** - The type of keystore that Oxygen XML Developer plugin uses (JKS or PKCS-12).
- **Keystore file** - The location of the imported file.
- **Keystore password** - The password that is used for protecting the privacy of the stored keys.
- **Certificate alias** - The alias used for storing the key entry (the certificate or the private key) inside the keystore.
- **Private key password** - The private key password of the certificate (required only for JKS keystores).
- **Validate** - Press this button to verify the configured keystore and the validity of the certificate.

**XML Structure Outline Preferences**

To configure the XML Structure Outline options, open the Preferences dialog box and go to XML Structure Outline, which contains the following preferences:

- **Preferred attribute names for display** - The preferred attribute names when displaying the attributes of an element in the Outline view. If there is no preferred attribute name specified, the first attribute of an element is displayed.
- **Enable outline drag and drop** - Drag and drop is disabled for the tree displayed in the Outline view only if there is a possibility to accidentally change the structure of the document by such operations.

**Importing / Exporting Global Options**

The import/export buttons are located in the preferences page of the Oxygen XML Developer plugin. To open this page, go to Open the Preferences dialog box. You can use the import/export buttons to load or save global preferences as an XML file which can be reloaded both on your computer and on others.

The following actions are available:

- **Reset Global Options**
  - Restores the preference to the factory defaults, or to the custom defaults, if they are defined.
**Import Global Options**

Allows you to import a set of Global Options from an exported XML options file. You can also select a project file (.xpr) to import all the Global Options that are set in that project file. After you select a file the Import Global Options dialog box is displayed and it informs you that the operation will only override the options that are included in the imported file. You can enable the **Reset all other options to their default values** option to reset all options to the default values before the file is imported.

**Export Global Options**

Allows you to export Global Options to an XML options file. Some user-specific options that are private are not included. For example, the name of the Review Author is not included in the export operation.

**Reset Global Options**

To reset all global preferences to their default values, open the Preferences dialog box and click on Reset Global Options.

The list of transformation scenarios will be reset to the default scenarios.

**Customizing Default Options**

Oxygen XML Developer plugin has an extensive set of options that you can configure. When Oxygen XML Developer plugin is installed, these options are set to default values. You can provide a different set of default values for an installation using an options file.

**Creating an options file**

To create an options file:

1. You may wish to use a fresh install for this procedure, to make sure that you do not copy personal option settings to the group.
2. Open Oxygen XML Developer plugin and open the Preferences dialog box.
3. Go through the options and set them to the desired defaults. Make sure that you are setting global options, not project options in each page.
4. Go to back to the main preferences page and click Export Global Options to create an options file.

**Providing Default Option Values**

Use either one of the following ways to configure an Oxygen XML Developer plugin installation to use customized default options from an XML configuration file:

- Set the path to the options file as the value of the `com.oxygenxml.default.options` system property.

  You can add the following line in the [Eclipse-platform-install-folder]/configuration/config.ini file:

  ```ini
  com.oxygenxml.default.options=file:\@config.dir/..//default-options.xml
  ```

- In the [OXYGEN_DIR] installation folder, create a folder called preferences. Copy the options file in the [Eclipse-platform-install-folder]/plugins/com.oxygenxml.editor/preferences folder or to the equivalent plugin folder in the [Eclipse-platform-install-folder]/dropins folder if the plugin was installed as a drop-in.

  **Note:** Make sure that the options configuration file has the `.xml` extension (for example: `default-options.xml`).
Scenarios Management

You can import and export the global transformation and validation scenarios using the following actions:

- To load a set of transformation scenarios from a properties file, open the Preferences dialog box and go to Scenarios Management > Import Global Transformation Scenarios.
- To store a set of transformation scenarios in a properties file, open the Preferences dialog box and go to Scenarios Management > Export Global Transformation Scenarios.
- To load a set of validation scenarios from a properties file, open the Preferences dialog box and go to Scenarios Management > Import Global Validation Scenarios.
- To store all the global (not project-level) validation scenarios in a properties file, open the Preferences dialog box and go to Scenarios Management > Export Global Validation Scenarios.

The Export Global Transformation Scenarios and Export Global Validation Scenarios options are used to store all the scenarios in a separate properties file. Associations between document URLs and scenarios are also saved in this file. You can load the saved scenarios using the Import Global Transformation Scenarios and Import Global Validation Scenarios actions. To distinguish the existing scenarios and the imported ones, the names of the imported scenarios contain the word import.

Editor Variables

An editor variable is a shorthand notation for context-dependent information, such as a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example, the input URL of a transformation, the output file path of a transformation, or the command line of an external tool) to make a command or a parameter generic and re-usable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Developer plugin commands of external engines or other external tools, in transformation scenarios, and in validation scenarios:

- `${oxygenHome}` - Oxygen XML Developer plugin installation folder as URL.
- `${oxygenInstallDir}` - Oxygen XML Developer plugin installation folder as file path.
- `${framework}` - The path (as URL) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- `${framework(fr_name)}` - The path (as URL) of the `fr_name` framework.
- `${frameworkDir(fr_name)}` - The path (as file path) of the `fr_name` framework.

**Note:** Because multiple frameworks might have the same name (although it is not recommended), for both `${framework(fr_name)}` and `${frameworkDir(fr_name)}` editor variables Oxygen XML Developer plugin employs the following algorithm when searching for a given framework name:

- all frameworks are sorted, from high to low, according to their Priority setting from the Document Type Association preferences page. Only frameworks that have the Enabled checkbox set are taken into account.
- next, if the two or more frameworks have the same name and priority, a further sorting based on the Storage setting is made, in the exact following order:
  - frameworks stored in the internal Oxygen XML Developer plugin options
  - additional frameworks added in the Locations preferences page
  - frameworks installed using the add-ons support
  - frameworks found in the main frameworks location (Default or Custom)

- `${frameworks}` - The path (as URL) of the `[OXYGEN_DIR]` directory.
- The path (as file path) of the current framework, as part of the [OXYGEN_DIR]/frameworks directory.
- The path (as file path) of the current framework, as part of the [OXYGEN_DIR]/frameworks directory.
- The path (as URL) of the user home folder.
- The path (as file path) of the user home folder.
- Current project folder as URL. Usually the current folder selected in the Project View.
- Current project folder as file path. Usually the current folder selected in the Project View.
- Current project name.
- 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.
- Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- Current file name with extension. The current file is the one currently opened and selected.
- Current file as file path, that is the absolute file path of the current edited document.
- The local file path of the ZIP archive that includes the current edited document.
- The URL path of the ZIP archive that includes the current edited document.
- The local directory path of the ZIP archive that includes the current edited document.
- The URL path of the directory of the ZIP archive that includes the current edited document.
- The file name (without parent directory and without file extension) of the zip archive that includes the current edited file.
- The file name (with file extension, for example .zip or .epub, but without parent directory) of the zip archive that includes the current edited file.
- Current file as URL, that is the absolute file path of the current edited document represented as URL.
- Path separator, that is the separator which can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- 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.
- The position where the caret is inserted. This variable can be used in a code template, in Author operations, or in a selection plugin.
- The current selected text content in the current edited document. This variable can be used in a code template, in Author operations, or in a selection plugin.
- Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java UUID class.
- Value of the VAR_NAME environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the $system(var.name) editor variable.
- Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as -Dvar.name=var.value. If you are looking for operating system environment variables, use the $env(VAR_NAME) editor variable instead.
- To prompt for values at runtime, use the ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value') - To prompt for values at runtime, use the ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value') editor variable. You can set the following parameters:
  - 'message' - The displayed message. Note the quotes that enclose the message.
  - type - Optional parameter, with one of the following values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>${ask('message', url, 'default_value')}</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parameter</td>
<td>Input is considered a URL. Oxygen XML Developer plugin checks that the provided URL is valid.</td>
</tr>
<tr>
<td>password</td>
<td>The input is hidden with bullet characters.</td>
</tr>
<tr>
<td>generic</td>
<td>The input is considered to be generic text that requires no special handling.</td>
</tr>
<tr>
<td>relative_url</td>
<td>Input is considered a URL. Oxygen XML Developer plugin tries to make the URL relative to that of the document you are editing.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>combobox</td>
<td>Displays a dialog box that offers a drop-down list. The drop-down list is populated with the given rendered_value values. Choosing such a value will return its associated value (real_value).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| editable_combobox | **Format:** \${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default'})

**Description:** Displays a dialog box that offers a drop-down list with editable elements. The drop-down list is populated with the given rendered_value values. Choosing such a value will return its associated real value (real_value) or the value inserted when you edit a list entry.

**Example:**
• \${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}
  - The dialog box has the name 'Operating System'. The drop-down list displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.

| radio | **Format:** \${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}

**Description:** Displays a dialog box that offers a series of radio buttons. Each radio button displays a rendered_value and will return an associated real_value.

**Example:**
• \${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}
  - The dialog box has the name 'Operating System'. The radio button group allows you to choose between the three operating systems.

  **Note:** In this example Mac OS X is the default selected value and if selected it would return osx for the output.

- 'default-value' - optional parameter. Provides a default value.
- \${date(pattern)} - Current date. The allowed patterns are equivalent to the ones in the Java SimpleDateFormat class.
  **Example:** yyyy-MM-dd;

  **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to [http://www.w3.org/TR/xmllschema-2/#date](http://www.w3.org/TR/xmllschema-2/#date). For details about xs:datetime, go to [http://www.w3.org/TR/xmlschema-2/#dateTime](http://www.w3.org/TR/xmlschema-2/#dateTime).

- \${dbgXML} - The local file path to the XML document which is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of an external tool or of a custom validator, the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

You can configure the custom editor variables in the Custom Editor Variables preferences page.

Localization of the User Interface

To localize the Oxygen XML Developer plugin, you can use one of the following methods:

• localization through the update site:

• localization through the zip archive:
  Go to http://www.oxygenxml.com/download.html and download the zip archive with the plugin language pack. Unzip the downloaded zip archive in the dropins subdirectory of the Eclipse install directory. Restart Eclipse.
If your operating system is running in the language you want to start Eclipse in (for example, you are using Japanese version of Windows, and you want to start Eclipse in Japanese), Oxygen XML Developer plugin matches the appropriate language from the language pack. However, if your operating system is running in a language other than the one you want to start Eclipse in (for example, you are using the English version of Windows, and you want to start Eclipse in Japanese, if you have the required operating system language support including the keyboard layouts and input method editors installed), specify the `-nl <locale>` command line argument when you launch Eclipse. Oxygen XML Developer plugin uses the translation file which matches the specified `<locale>`.

You can also localize the Eclipse plugin to a different language that the initial languages in the language pack. Duplicate the `plugin.properties` file from the Oxygen XML Developer plugin plugin installation directory, translate all the keys in the file and change its name to `plugin_<locale>.properties`. 
Chapter 17

Common Problems

Topics:  This section lists the most commonly found problems and their solutions.

- Performance Problems
- Common Problems and Solutions
Performance Problems

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

External Processes

The *Memory available to the built-in FOP* option controls the amount of memory allocated to generate PDF output with the built-in Apache FOP processor. If Oxygen XML Developer plugin throws an *Out Of Memory* error, open the *Preferences* dialog box, go to XML > XSLT-FO-XQuery > FO Processors, and increase the value of the *Memory available to the built-in FOP* option.

For external XSL-FO processors, XSLT processors, and external tools, the maximum value of the allocated memory is set in the command line of the tool using the `-Xmx` parameter set to the Java virtual machine.

Common Problems and Solutions

This chapter presents common problems that may appear when running the application and the solutions for these problems.

Oxygen XML Developer plugin Takes Several Minutes to Start on Mac

If Oxygen XML Developer plugin takes several minutes to start, the Java framework installed on the Mac may have a problem. One solution for this is to update Java to the latest version: go to Apple symbol > Software Update. After it finishes to check for updates, click *Show Details*, select the Java Update (if one is available) and click *Install*. If no Java updates are available, reset the Java preferences to their defaults. Start Applications > Utilities > Java Preferences and click *Restore Defaults*.

XSLT Debugger Is Very Slow

When I run a transformation in the XSLT Debugger perspective it is very slow. Can I increase the speed?

If the transformation produces HTML or XHTML output you should disable rendering of output in the XHTML output view during the transformation process. To view the XHTML output result do one of the following:

- Run the transformation in the Editor perspective and make sure the *Open in Browser/System Application* option is enabled.
- Run the transformation in the XSLT Debugger perspective, save the text output area to a file, and use a browser application for viewing it (for example Firefox or Internet Explorer).

Syntax Highlight Not Available in Eclipse Plugin

I associated the .ext extension with Oxygen XML Developer plugin in Eclipse. Why does an .ext file opened with the Oxygen XML Developer plugin not have syntax highlight?

Associating an extension with Oxygen XML Developer plugin in Eclipse 3.7 requires three steps:

1. Associate the .ext extension with the Oxygen XML Developer plugin.
   a) *Open the Preferences* dialog box and go to General > Editors > File Associations.
   b) Add *.*ext to the list of file types.
   c) Select *.*ext in the list by clicking on it.
   d) Add Oxygen XML Developer plugin to the list of *Associated editors* and make it the default editor.

2. Associate the .ext extension with the Oxygen XML content type.
   a) *Open the Preferences* dialog box and go to General > Content Types.
   b) Add *.*ext to the *File associations* list for the Text > XML > oXygen XML content type.

3. Press the OK button in the Eclipse preferences dialog box.
Now when an *.ext. file is opened the icon of the editor and the syntax highlight should be the same as for XML files opened with the Oxygen XML Developer plugin.

**Damaged File Associations on OS X**

After upgrading OS X and Oxygen XML Developer plugin, it is no longer associated to the appropriate file types (such as XML, XSL, XSD, etc.) How can I create the file associations again?

The upgrade damaged the file associations in the LaunchService Database on your OS X machine. Please rebuild the LaunchService Database with the following procedure. This will reset all file associations and will rescan the entire file system searching for applications that declare file associations and collecting them in a database used by Finder.

1. Find all the Oxygen XML Developer plugin installations on your hard drive.
2. Delete them by dragging them to the Trash.
3. Clear the Trash.
4. Unpack the Oxygen XML Developer plugin 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:

   ```bash
   /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/LaunchServices.framework/Versions/A/Support/lsregister -kill -r -domain local -domain system -domain user
   ```

7. Restart Finder with the following command:

   ```bash
   killall Finder
   ```

8. Create a XML or XSD file on your desktop.
   It should have the Oxygen XML Developer plugin icon.


10. Accept the confirmation.

When you start Oxygen XML Developer plugin the file associations should work correctly.

**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:

```properties
log4j.rootCategory= debug, R2
log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error and close the application. The log file is called `logging.log` and is located in the install folder.

**Signature verification failed error on open or edit a resource from Documentum**

When I try to open/edit a resource from Documentum, I receive the following error:

```
signature verification failed: certificate for All-MB.jar.checksum not signed by a certification authority.
```
The problem is that the certificates from the Java Runtime Environment 1.6.0_22 or later no longer validate the signatures of the UCF jars.

Edit the `eclipse.ini` file from the Eclipse directory and add the following parameter to the `-vmargs`:

```
-Drequire.signed.ucf.jars=false, for example:
```

```
-vmargs
-Xms40m
-Xmx256m
-Drequire.signed.ucf.jars=false
```

**Compatibility Issue Between Java and Certain Graphics Card Drivers**

Under certain settings, a compatibility issue can appear between Java and some graphics card drivers, which results in the text from the editor (in Author or Text mode) being displayed garbled. In case you encounter this problem, update your graphics card driver. Another possible workaround is, open the Preferences dialog box, go to Fonts > Text antialiasing, and set the value of Text antialiasing option to ON.

**Note:** If this workaround does not resolve the problem, set the Text antialiasing option to other values.

**An Image Appears Stretched Out in the PDF Output**

Sometimes, when publishing XML content (DITA, DocBook, etc), images are scaled up in the PDF outputs but are displayed perfectly in the HTML (or WebHelp) output.

PDF output from XML content is obtained by first obtaining a intermediary XML format called XSL-FO and then applying an XSL-FO processor to it to obtain the PDF. This stretching problem is caused by the fact that all XSL-FO processors take into account the DPI (dots-per-inch) resolution when computing the size of the rendered image.

The PDF processor which comes out of the box with the application is the open-source Apache FOP processor. Here is what Apache FOP does when deciding the image size:

1. If the XSL-FO output contains width, height or a scale specified for the image `external-graphic` tag, then these dimensions are used. This means that if in the XML (DITA, DocBook, etc) you set explicit dimensions to the image they will be used as such in the PDF output.
2. If there are no sizes (width, height or scale) specified on the image XML element, the processor looks at the image resolution information available in the image content. If the image has such a resolution saved in it, the resolution will be used and combined with the image width and height in order to obtain the rendered image dimensions.
3. If the image does not contain resolution information inside, Apache FOP will look at the FOP configuration file for a default resolution. The FOP configuration file for XSLT transformations which output PDF is located in the `lib/fop.xconf`. DITA publishing uses the DITA Open Toolkit which has the Apache FOP configuration file located in `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf`. The configuration file contains two XML elements called `source-resolution` and `target-resolution`. The values set to those elements can be increased, usually a DPI value of 110 or 120 should render the image in PDF just like in the HTML output.

The commercial **RenderX XEP** XSL-FO processor behaves similarly but as a fallback it uses 120 as the DPI value instead of using a configuration file.

**Tip:**

As a conclusion, it is best to save your images without any DPI resolution information in them. For example the open-source GIMP image editor allows you when saving a PNG image whether to save the resolution to it or not:
Having images without any resolution information saved in them allows you to control the image resolution from the configuration file for all referenced images.

The DITA PDF Transformation Fails

To generate the PDF output, Oxygen XML Developer plugin uses the DITA Open Toolkit.

You can analyse the Results tab of the DITA transformation and search for messages that contain text similar to `[fop] [ERROR]`. If you encounter this type of error message, edit the transformation scenario you are using and set the `clean.temp` parameter to `no` and the `retain.topic.fo` parameter to `yes`. Run the transformation, go to the temporary directory of the transformation, open the `topic.fo` file and go to the line indicated by the error. Depending on the XSL FO context try to find the DITA topic that contains the text which generates the error.

If none of the above methods helps you, go to Help > About > Components > Frameworks and check what version of the DITA Open Toolkit you are using. Copy the whole output from the DITA OT console output and either report the problem on the DITA User List or to support@oxygenxml.com.

The DITA to CHM Transformation Fails

Oxygen XML Developer plugin uses the DITA Open Toolkit and the HTML Help compiler (part of the Microsoft HTML Help Workshop) to transform DITA content into Compiled HTML Help (or CHM in short).

However, the execution of the transformation scenario may still fail. Reported errors include:

- [exec] HHC5010: Error: Cannot open "fileName.chm". Compilation stopped. - this error occurs when the CHM output file is opened and the transformation scenario cannot rewrite its content. To solve this issue, close the CHM help file and execute the transformation scenario again.
- [exec] HHC5003: Error: Compilation failed while compiling fileName - possible causes of this error are:
  - the processed file does not exist. Fix the file reference before executing the transformation scenario again.
  - the processed file has a name that contains space characters. To solve the issue, remove any spacing from the file name and execute the transformation scenario again.
DITA Map ANT Transformation Because it Cannot Connect to External Location

The transformation is run as an external ANT process so you can continue using the application as the transformation unfolds. All output from the process appears in the DITA Transformation tab.

The HTTP proxy settings are used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the Network Connections.

Topic References outside the main DITA Map folder

Referencing to a DITA topic, map or to a binary resource (for example: image) which is located outside of the folder where the main DITA Map is located usually leads to problems when publishing the content using the DITA Open Toolkit. The DITA OT does not handle well links to topics which are outside the directory where the published DITA Map is found. By default it does not even copy the referenced topics to the output directory.

You have the following options:

1. Create another DITA Map which is located in a folder path above all referenced folders and reference from it the original DITA Map. Then transform this DITA Map instead.
2. Edit the transformation scenario and in the Parameters tab edit the fix.external.refs.com.oxygenxml parameter. This parameter is used to specify whether the application tries to fix up such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references. The fix up has no impact on your edited DITA content. Only "false" and "true" are valid values. The default value is false.

The PDF Processing Fails to Use the DITA OT and Apache FOP

There are cases when publishing DITA content fails when creating a PDF file. This topic lists some common problems and solutions.

- The FO processor cannot save the PDF at the specified target. The console output contains messages like:

```
[fop] [ERROR] AntTask - Error rendering fo file: C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo <Failed to open C:\samples\dita\out\pdf\test.pdf>
Failed to open samples\dita\out\pdf\test.pdf
..............
[fop] Caused by: java.io.FileNotFoundException: C:\Users\radu_coravu\Desktop\bev\out\pdf\test.pdf
(The process cannot access the file because it is being used by another process)
```

Such an error message usually means that the PDF file is already opened in a PDF reader application. The solution is to close the open PDF before running the transformation.

- One of the DITA tables contains more cells in a table row than the defined number of colspec elements. The console output contains messages like:

```
[fop] [ERROR] AntTask - Error rendering fo file: D:\projects\eXml\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo
<net.sf.saxon.trans.XPathException; org.apache.fop.fo.ValidationException:
The column-number or number of cells in the row overflows the number of fo:table-columns specified for the table. (See position 179:-1)>
<net.sf.saxon.trans.XPathException; org.apache.fop.fo.ValidationException: The column-number or number of cells in the row overflows the number of fo:table-columns specified for the table. (See position 179:-1)>
[fop] at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:657)
[fop] at net.sf.saxon.event.ContentHandlerProxy.startContent(ContentHandlerProxy.java:375)
..............
[fop] D:\projects\samples\dita\flowers\out\pdf\flowers.pdf
```

To resolve this issue, correct the colspec attribute on the table that caused the issue. To locate the table that caused the issue:

1. Edit the transformation scenario and set the parameter clean.temp to no.
2. Run the transformation, open the topic.fo file in Oxygen XML Developer plugin, and look in it at the line specified in the error message (See position 179:-1).
3. Look around that line in the XSL-FO file to find relevant text content which you can use, for example, with the Find/Replace in Files action in the DITA Maps Manager view to find the original DITA topic for which the table was generated.
There is a broken link in the generated XSL-FO file. The PDF is generated but contains a link that is not working. The console output contains messages like:

```
```

To resolve this issue:

1. Use the Validate and Check for Completeness action available in the DITA Maps Manager view to find such problems.
2. If you publish to PDF using a DITAVAL filter, select the same DITAVAL file in the DITA Map Completeness Check dialog box.
3. If the Validate and Check for Completeness action does not discover any issues, edit the transformation scenario and set the clean.temp parameter to no.
4. Run the transformation, open the topic.fo file in Oxygen XML Developer plugin, and search in it for the unique_4_Connect_42_wrongID id.
5. Look around that line in the XSL-FO file to find relevant text content which you can use, for example, with the Find/Replace in Files action in the DITA Maps Manager view to find the original DITA topic for which the table was generated.

The TocJS Transformation Doesn't Generate All Files for a Tree-Like TOC

The TocJS transformation of a DITA map does not generate all the files needed to display the tree-like table of contents. To get a complete working set of output files you should follow these steps:

1. Run the XHTML transformation on the same DITA map. Make sure the output gets generated in the same output folder as for the TocJS transformation.
2. Copy the content of [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/basefiles folder in the transformation's output folder.
5. Locate element <frame name="contentwin" src="concepts/about.html">.
6. Replace "concepts/about.html" with "index.html".

Navigation to the web page was canceled when viewing CHM on a Network Drive

When viewing a CHM on a network drive, if you only see the TOC and an empty page displaying “Navigation to the web page was canceled” note that this is normal behaviour. The Microsoft viewer for CHM does not display the topics for a CHM opened on a network drive.

As a workaround, copy the CHM file on your local system and view it there.

Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x

On some Linux systems based on Gnome 3.x (e.g. Ubuntu 11.x, 12.x) the main menu of Oxygen XML Developer plugin has alignment issues when you navigate it using your mouse.

This is a known problem caused by Java SE 6 1.6.0_32 and earlier. You can resolve this problem using the latest Java SE 6 JRE from Oracle. To download the latest version, go to http://www.oracle.com/technetwork/java/javase/downloads/index.html.

To bypass the JRE bundled with Oxygen XML Developer plugin, go to the installation directory of Oxygen XML Developer plugin and rename or move the jre folder. If Oxygen XML Developer plugin does not seem to locate the system JRE, either set the JAVA_HOME environment variable to point to the location where you have installed the
JRE, or you can simply copy that folder with the JRE to the installation directory and rename it to `jre` to take the place of the bundled JRE.

**SVG Rendering Issues**

Oxygen XML Developer plugin uses the Apache Batik open source library to render SVG images. The Batik library only has partial support for SVG 1.2: [http://xmlgraphics.apache.org/batik/dev/svg12.html](http://xmlgraphics.apache.org/batik/dev/svg12.html).

For example, if you are using the Inkscape SVG editor, it is possible that it saves the SVG as 1.1 but it actually uses SVG 1.2 elements like `flowRoot` inside it. This means that the image will not be properly rendered inside the application.

**MSXML 4.0 Transformation Issues**

In case the latest MSXML 4.0 service pack is not installed on your computer, you are likely to encounter the following error message in the **Results** panel when you run a transformation scenario that uses the MSXML 4.0 transformer.

```
Error Message
Could not create the 'MSXML2.DOMDocument.4.0' object.
Make sure that MSXML version 4.0 is correctly installed on the machine.
```

To fix this issue, go to the Microsoft website and get the latest MSXML 4.0 service pack.
Java Archive

JAR (Java ARchive) is an archive file format. JAR files are built on the ZIP file format and have the .jar file extension. Computer users can create or extract JAR files using the `jar` command that comes with a JDK.

Java Archive (JAR)

Apache Ant

Apache Ant (Another Neat Tool) is a software tool for automating software build processes.

Ant

Active cell

The selected cell in which data is entered when you begin typing. Only one cell is active at a time. The active cell is bounded by a heavy border.

Block element

A block element is an one that is intended to be visually separated from its siblings, usually vertically. For instance, a paragraph or a list item are block elements. It is distinct from a inline element which has no such separation.

Inline element

An inline element is one that is intended to be displayed in the same line of text as its siblings or the surrounding text. For instance, strong and emphasis in HTML are inline elements. It is distinct from a block element, which is visually separated from its siblings.

DITA map

A DITA map is a hierarchical collection of DITA topics that can be processed to form an output. Maps do not contain the content of topics, but only references to them. These are known as topic references. Usually the maps are saved on disk or in a CMS with the extension `.ditamap`.

Maps can also contain relationship tables that establish relationships between the topics contained within the map. Relationship tables are also used to generate links in your published document.

You can use your map or bookmap to generate a deliverable using an output type such as XHTML, PDF, HTML Help or Eclipse Help.
Bookmap

A bookmap is a specialized ditamap used for creating books. A bookmap supports book divisions such as chapters and book lists such as indexes.
Index

A
Archives 403, 404, 407
browse 404
edit 407
file browser 404
modify 404

B
Bidirectional text 51
Grid Mode 51

C
command line 359
Common Problems 552
Configuration 487
CSS validator 487
Configure Application 517
Editor preferences 517
spell check 517
Configure the Application 486, 487, 488, 491, 492, 502, 503, 504,
505, 506, 507, 509, 512, 513, 517, 519, 521, 522, 523,
524, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535,
536, 539, 540, 541, 542, 543, 544
(S)FTP 522
archive 487
certificates 542
customize default options 543
custom validation 519
data sources 488, 491, 492
download links for database drivers 491
table filters 492
document type association 492
document checking 519
deeditor variables 544
fonts 521
HTTP(S)/WebDAV preferences 521

Configure the Application (continued)
import 540, 541
date/time patterns 541
import/export global options 542
license 486
outline 542
reset global options 543
scenarios management 523, 544
views 523
XML 523
XML catalog 523
XML instances generator 526
XML parser 524
XProc engines 527
XSLT 528
XSLT/FO/XQuery 528
XSLT/FO/XQuery preferences 528, 529, 530, 531, 532, 533,
534, 535, 536, 539
custom engines 539
debugger 535
FO Processors 536
MSXML 532
MSXML.NET 532
profiler 536
Saxon6 528
Saxon HE/PE/EE 529, 534
Saxon-HE/PE/EE 529
Saxon HE/PE/EE advanced options 530, 535
XPath 539
XQuery 533
XSLTProc 531
Content Management System 467
Copy/Paste 49
gird editor 49

D
Databases 375, 409, 410, 430, 431, 449, 450, 452, 453, 473
debugging with MarkLogic 452
limitations of the MarkLogic debugger 452
native XML databases (NXD) 430
Native XML databases (NXD) 431
Relational databases 410
SharePoint connection 473
WebDAV connection 453
XQuery 375, 449, 450, 452
debugging 452
drag and drop from the Data Source Explorer 449
transformation 450
validation 375
Debugging XSLT/XQuery Documents 382, 385, 394, 397
Java extensions 397
layout 382, 385, 394
information views 385
multiple output documents in XSLT 2.0 394
XSLT/XQuery debugger 394
Editing WSDL Document 229
SOAP request 229
composing a SOAP request 229

Editing WSDL Documents 214, 217, 218, 219, 220, 223, 224, 225, 229, 231
Component Dependencies view 223
component occurrences 224
composing web service calls with WSDL SOAP analyzer 229
content completion 217
contextual editing 218
generate documentation for WSDL documents 225
generate documentation for WSDL documents from command line 229
generate documentation for WSDL documents in a custom format 229
Outline view 214
Quick Assist 225
Resource Hierarchy/Dependencies view 220
searching and refactoring operations 219
searching and refactoring operations scope 220
SOAP request 231
testing remote WSDL files 231
UDDI registry browser 231

Editing XML Documents 64, 71, 72, 74, 78, 79, 80, 81, 83, 84, 85, 86, 88, 90, 91, 92, 95, 96, 97, 99, 101, 110, 111, 112, 113, 114
against a schema 84
associate a schema to a document 71, 72, 74
add schema association in XML instance 72
learning a document structure 74
setting a default schema 71
supported schema types 71
checking XML well-formedness 83
code templates 81
content completion 81
converting between schema languages 101
document navigation 91, 92
folding 91
outline view 92
editor specific actions 110, 111, 112, 113, 114
document actions 112
edit actions 110
refactoring actions 113
select actions 111
smart editing 113
source actions 111
syntax highlight depending on namespace prefix 114
grouping documents in XML projects 64, 95
large documents 95
new project 64
project view 64
including document parts with XInclude 96
Resource Hierarchy/Dependencies view 99
status information 110
streamline with content completion 74, 78, 79, 80
the Annotation panel 79
the Attributes view 79
the Elements view 80
the Entities view 80
the Model panel 78
validation against a schema 84, 85, 86, 88, 90, 91
automatic validation 85

Editing XML Documents (continued)
validation against a schema (continued)
custom validation 86
marking validation errors 84
resolving references to remote schemas with an XML Catalog 91
validation actions 90
validation example 85
validation scenario 88
working with XML Catalogs 97

Editing XML Schemas 155, 189, 190, 192, 195, 197, 200, 201, 204, 208, 209
Component Dependencies view 190
contextual editing 189
generate documentation for XML Schema 195, 197, 200, 201
as HTML 197
as PDF, DocBook or custom format 200
from command line 201
relational database table to XML schema 209
Resource Hierarchy/Dependencies view 192
schema instance generator 204
schema regular expressions builder 208
searching and refactoring actions 189

Editing XProc Scripts 259
Editing XQuery Documents 211, 212, 213
folding 212
generate HTML documentation 213
Editing XSL Stylesheets 150, 151, 154
Component Dependencies view 150
quick assist support 151
XSpec 154
Editing XSLT Schemas 128
contextual editing 128
Editing XSLT Stylesheets 127, 128, 129, 130, 134, 135, 137, 138, 140, 143, 144, 145, 147
content completion 129, 130
in XPath expressions 130
find XSLT references and declarations 144
generate documentation for XSLT stylesheets 138
generate documentation for XSLT Stylesheets 140, 143
as HTML 140
from command line 143
in custom format 143
Outline view 135
refactoring actions 145
Resource Hierarchy/Dependencies view 147
validation 127, 128
custom validation 128
validation scenario 128
XSLT Input view 134
XSLT stylesheet documentation 137
EPUB Document Type 305
external process 359

F
Find/Replace 44
Find All Elements/Attributes dialog box 44
Format and indent 103
Format and Indent 508
G

Getting Started 35, 36, 40, 41
perspectives 36, 40, 41
database 41
editor 36
XQuery debugger 40
XSLT debugger 40
grid editor 47, 48
navigation 47, 48
collapse all 48
collapse children 48
collapse others 48
expand all 47
expand children 48
Grid Editor 45, 46, 47, 48, 49
add nodes 48
clear column content 48
copy/paste 49
drag and drop 49
duplicate nodes 48
inserting table column 48
insert table row 48
layouts (grid and tree) 46
navigation 47
refresh layout 49
sort table column 48
start and stop editing a cell value 49

I

Importing Data 458
Importing Data 458, 461, 463
from a database 458
table content as XML document 458
from database 461
convert table structure to XML Schema 461
from HTML files 463
from MS Excel 461
from text files 463
Installation 21, 22, 23
Eclipse 21, 22, 23
update site method (Eclipse 3.6 - 4.5) 21, 22, 23
ZIP archive method (Eclipse 3.6 - 4.5) 21, 22, 23

J

JATS NISO Journal Article Tag Suite Document Type 305

L

License 24, 25, 26, 27, 28, 30, 31, 32
floating (concurrent) license 26
floating license server 30
floating license servlet 28
license server installed on OS X Linux Unix 31
multiple named-user licenses 26
named-user license 25
register a license key 24
release floating license 27
releasing a license key 32
transferring a license key 32

License (continued)
unregistering a license key 32

M

Master Files 69

N

Native XML Databases (NXD) 422, 430, 431, 432, 433, 434, 435
database connections configuration 431, 432, 434
Berkeley DB XML 431
Documentum xDb (X-Hive/DB) 434
eXist 432
data sources configuration 430, 431, 432, 433, 434
Berkeley DB XML 431
Documentum xDb (X-Hive/DB) 434
eXist 432
MarkLogic 433
resource management 422, 435
Data Source Explorer view 422, 435

O

OutOfMemory 506, 512, 536, 552
Out Of Memory 506, 512, 536, 552
OutOfMemoryError 506, 512, 536, 552
outside oxygen 359

P

Performance Problems 552
external processes 552
Preferences 486
Pretty print 103
Profiling 400
XSLT stylesheets and XQuery documents 400
Profiling XSLT Stylesheets and XQuery Documents 400, 401
profiling information 400
Hotspots view 400
Invocation tree view 400
XSLT/XQuery profiler 401

Q

Querying Documents 211, 368, 372, 373, 374, 375, 376
running XPath and XQuery expressions 368
XPath/XQuery Builder view 368
running XPath expressions 368
XQuery 211, 372, 373, 374, 375, 376
Input view 374
other editing actions 376
Outline view 211, 373
syntax highlight and content completion 372
transforming XML documents; advanced Saxon B/SA options 376
validation 375
Relational Databases 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 421, 422, 426, 428, 429, 430, 435, 458, 461
connections configuration 410, 411, 413, 415, 416, 418, 421, 422
generic JDBC 415
IBM DB2 connection 411
JDBC-ODBC connection 422
Microsoft SQL Server 413
MySQL 416
Oracle 11g 418
PostgreSQL 8.3 421
creating XML Schema from databases 461
data sources configuration 410, 411, 412, 414, 415, 417, 419
generic JDBC data source 414
IBM DB2 411
Microsoft SQL Server 412
MySQL 415
Oracle 11g 417
PostgreSQL 8.3 419
importing from databases 458
resource management 422, 426, 435
Data Source Explorer view 422, 435
Table Explorer view 426
SQL execution support 428, 429, 430
drag and drop from the Data Source Explorer 428
executing SQL statements 430
SQL validation 429
Relax NG Schema Editor 236
contextual editing 236

S
Schematron Quick Fix 115
SharePoint Connection 473, 474, 475
actions at connection level 474
actions at file level 475
actions at folder level 474
configuration 473
SQF 115

T
TEI ODD document type 303
association rules 303
Author extensions 303
catalogs 303
schema 303
TEI ODD Document Type 303
Author extensions 303
templates 303
transformation scenarios 303
TEI P4 document type 303
association rules 303
Author extensions 303
catalogs 303
schema 303
TEI P4 Document Type 303, 304
Author extensions 304
templates 304
transformation scenarios 304
TEI P5 document type 304
association rules 304
Author extensions 304
catalogs 304
schema 304
TEI P5 Document Type 304
Author extensions 304
templates 304
transformation scenarios 304
Text Editing Mode 44
Tools 477
Transformation Scenario 308, 309, 310, 311, 312, 314, 336, 337
built-in transformation scenarios 337
new transformation scenario 308, 309, 310, 311, 312, 314, 336
additional XSLT stylesheets 312
configure transformation scenario 308
create a transformation scenario 336
XML transformation with XSLT 309
XQuery parameters 310
XSLT/XQuery extensions 311, 314
XSLT parameters 310
Transforming Documents 307, 308, 337, 340, 342, 346
custom XSLT processors 342
output formats 346
supported XSLT processors 340
transformation scenario 308
Transformation Scenarios view 337
XSL-FO processors 342
XSLT processors extensions paths 342

U
Unicode 508
Uninstalling the Plugin 34
Upgrade 33
check for new version 33

V
Validating XML Documents 82

W
WebDAV Connection 453, 454
actions at connection level 453
actions at file level 454
actions at folder level 454
configuration 453
WebHelp 359
WebHelp Internationalization 299
WebHelp localization 299
WebHelp i18n 299
whitespace 508
Whitespace handling 103

X
XHTML document type 302
association rules 302
Author extensions 302
catalogs 302
XHTML document type (continued)
schema 302
XHTML Document Type 302
Author extensions 302
templates 302
transformation scenarios 302
XML Outline View 92, 93, 94, 95
document structure change 94
contextual menu 94
document tag selection 95
modification follow-up 94
outline filters 93
XML document overview 93
XML Quick Fixes 115
XML Schema Diagram Editor 155, 157, 158, 161, 162, 163, 164, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 183, 184, 186, 187, 188
Attributes view 184
inge schema namespaces 188
edit schema namespaces 188
Facets view 186, 187
inge patterns 187
group schema components 174
attributes 174
constraints 174
substitutions 174
navigation 175
Outline view 183
schema components 157, 158, 161, 162, 163, 164, 166, 167, 168, 169, 170, 171, 172, 173
xs:alternative 166
xs:any 169
xs:anyAttribute 170
xs:assert 172
xs:attribute 161
xs:attributeGroup 162
xs:complexType 163
xs:element 158
xs:field 172
xs:group 167
xs:import 167
xs:include 167
xs:key 171

XML Schema Diagram Editor (continued)
schema components (continued)
xs:keyRef 171
xs:notation 168
xs:openContent 173
xs:override 168
xs:redefine 168
xs:schema 157
xs:selector 172
xs:sequence, xs:choice, xs:all 169
xs:simpleType 164
xs:unique 171
the Palette view 187
validation 175
XML Schema Text Editor 188, 202
content completion 188
flatten an XML Schema 202
XML serialization 103
XQJ Connection 377
XQJ configuration 377
XQJ Support 377
XQJ processor configuration 377
XSLT/XQuery Debugger 385, 386, 387, 388, 389, 390, 391, 392, 394, 395
depbug steps 394
determining what XSLT/XQuery expression generated particular output 395
using breakpoints 395
inserting breakpoints 395
removing breakpoints 395
viewing processing information 385, 386, 387, 388, 389, 390, 391, 392
breakpoints view 387
context node view 385
messages view 388
node set view 392
output mapping stack view 389
stack view 388
templates view 391
variables view 392
XPath watch view 386